Refine Search

Search Results -

Term	Documents
(24 AND 5).USPT.	13
(L5 AND L24).USPT.	13

Database:

- US Pre-Grant Publication Full-Text Database
- US Patents Full-Text Database
- US OCR Full-Text Database
- EPO Abstracts Database
- JPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Search: L25

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Sunday, May 30, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L25	l5 and L24	13	L25
L24	l4.ab.	19	L24
L23	l20 and L22	20	L23
L22	l2 same l5	3191	L22
L21	l2 same l5L20	0	L21
L20	l14 and L19	44	L20
L19	bus and l4	524	L19
L18	l5 and L17	4	L18
L17	l15 and L16	4	L17
L16	l4 same l5	122	L16
L15	l13 and L14	239	L15
L14	l2 near1 data$1	8031	L14
L13	interrupt process$	7795	L13

n.2 2nd

real time Analog data

rela time data

h e b b cg b e e ch

L12	l10 and L11	11	L12
L11	l3 near3 CPU	8475	L11
L10	l6 and L9	30	L10
L9	(non) near1 l2	2835	L9
L8	l6 and L7	11	L8
L7	(no) near1 l2	631	L7
L6	l4 and L5	374	L6
L5	priorit$	147862	L5
L4	l2 near1 L3	711	L4
L3	interrupt$	306443	L3
L2	real time$1	99254	L2
L1	606839.pn.	0	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Term	Documents
(13 AND 22).USPT.	1
(L13 AND L22).USPT.	1

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L23

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, June 02, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L23	l13 and L22	1	L23
L22	context	173949	L22
L21	l13 and L20	0	L21
L20	non near1 (realtime$1 or real time$1)	3000	L20
L19	l4 and l13	1	L19
L18	l3 and L17	1	L18
L17	l13 and L16	1	L17
L16	generat$ and interrupt$	180096	L16
L15	l13 and event$1	0	L15
L14	l9 and L13	1	L14
L13	6295574.pn.	1	L13
L12	l4 and l10	1	L12
L11	l9 and L10	0	L11

h e b b cg b e e ch

L10	5603035.pn.	1	L10
L9	operating system	63989	L9
L8	l6 and L7	17	L8
L7	interrupt$.ab.	24836	L7
L6	l3 and L4	66	L6
L5	l3 and L4	66	L5
L4	priorit$	148408	L4
L3	l1 and L2	125	L3
L2	real time near3 (analog data or signal or peripheral device)	10166	L2
L1	real time near interrupt$	712	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

L16: Entry 2 of 7 File: USPT Jun 9, 1998

DOCUMENT-IDENTIFIER: US 5764852 A
TITLE: Method and apparatus for speech recognition for distinguishing non-speech audio input events from speech audio input events

Abstract Text (1):
A method and apparatus for analyzing audio input events. A template is utilized to analyze audio input events. A speech audio input event is identified. The identified speech audio input event is recorded. The recorded speech audio input event is processed to create a first entry in a template. A selected non-speech audio input event which occurs in a selected environment is identified. The identified non-speech audio input event is recorded. Then the recorded non-speech audio input event is processed to create a second entry in the template. Thereafter, a speech audio input event and a non-speech audio input event is distinguished by comparing an audio input event to the template.

Brief Summary Text (13):
The present invention provides method and apparatus for analyzing audio input events. The present invention utilizes a template to analyze audio input events. A speech audio input event is identified. The identified speech audio input event is recorded. The recorded speech audio input event is processed to create a first entry in a template. A selected non-speech audio input event which occurs in a selected environment is identified. The identified non-speech audio input event is recorded. Then the recorded non-speech audio input event is processed to create a second entry in the template. Thereafter, a speech audio input event and a non-speech audio input event is distinguished from each other by comparing an audio input event to the template, wherein the non-speech audio input event is identified.

Drawing Description Text (8):
FIG. 6 depicts a flow chart of a process for training and updating a voice recognition system to detect and differentiate between sounds that are speech audio input events or non-speech audio input event; and

Drawing Description Text (9):
FIG. 7 is a flowchart of a process for analyzing audio input events in a data processing system in accordance with a preferred embodiment of the present invention.

Detailed Description Text (3):
For example, the operation of CD-ROM player 17 may be controlled by multimedia application software which is resident in, and executed by, computer 15. The real-time digital data stream generated as an output of CD-ROM player 17 may be received and processed by computer 15 in accordance with instructions of the multimedia application resident therein. For example, the real-time digital data stream may be compressed for storage on a conventional computer floppy disk or for transmission via modem over ordinary telephone lines for receipt by a remotely located computer system which may decompress and play the digital streamed data on analog audio equipment. Alternatively, the real-time data stream output from CD-ROM player 17 may be received by computer 15, and subjected to digital or analog filtering, amplification, and sound balancing before being directed, in analog signal form, to

analog stereo amplifier 29 for output on audio speakers 31 and 33.

Detailed Description Text (4):
Microphone 19 may be used to receive analog input signals corresponding to ambient sounds. The real-time analog data stream may be directed to computer 15, converted into digital form, and subject to manipulation by the multimedia application software, such as a voice recognition program. The digital data may be stored, compressed, encrypted, filtered, subjected to transforms, outputted in analog form to analog stereo amplifier 29, directed as an output in analog form to telephone 23, presented in digitized analog form as an output of a modem for transmission on telephone lines, transformed into visual images for display on video monitor 25, or subjected to a variety of other different and conventional multimedia digital signal processing operations.

Detailed Description Text (5):
In a similar fashion, the analog and digital inputs and outputs of keyboard 21, telephone 23, and video monitor 25 may be subjected to conventional multimedia operations in computer 15. In particular, computer 15 may be used as a voice recognition system to direct commands and functions for other applications executing on computer 15. Microphone 19 may be used to receive speech audio input events, i.e., human speech, the audio input events may be processed using a multimedia application that is directed towards recognizing speech from analyzing inputs from microphone 19.

Detailed Description Text (6):
FIG. 2 is a block diagram representation of the principal hardware components which are utilized in the present invention to execute multimedia applications which control the operation of multimedia end devices 13. As is conventional in multimedia data processing operations, a central processing unit (CPU) 33 is provided in computer 15. Typically, the multimedia application software, such as a voice recognition application, is resident in RAM computer memory 35. CPU 33 executes the instructions which comprise the multimedia application. Also, as is typical in multimedia data processing operations, digital signal processor 37 is provided as an auxiliary processor, which is dedicated to performing operations on the real-time and/or asynchronous streamed data. As is well known to those skilled in the art, digital signal processors are microprocessor devices which are dedicated to performing operations based upon, or which include, real-time data and are thus designed to be very fast and respond quickly to allow the real-time operational nature of the multimedia end devices. Typically, in order to speed-up the operation of the digital signal processor 37, a conventional direct memory access (DMA) 39 is provided to allow for the rapid fetching and storing of data. In the present invention, separate instruction memory (IM) 41 and data memory (DM) 43 are provided to further speed up the operation of digital signal processor 37. Bus 45 is provided to communicate data between digital signal processor 37 and hardware interface 47, which includes digital-to-analog and analog-to-digital converters. Inputs and outputs for the various multimedia end devices 13 are connected through the digital-to-analog (D/A) and analog-to-digital (A/D) converter 47. In FIG. 2, a telephone input/output 49, a microphone input 53, and stereo outputs 55, 57 are depicted, in an exemplary manner, and are connected through the A/D and D/A converters in hardware interface 47. MIDI input/output also is connected to hardware interface 47 to digital signal processor 37 but is not connected to A/D or D/A converters.

Detailed Description Text (7):
Referring next to FIG. 3, a high level flow chart of a process employed by a user to train an application to recognize voice recognition commands is depicted in accordance with a preferred embodiment of the present invention. The user must "train" a set, i.e., a template of phrases for recognition, as illustrated in block 300. The user also defines a set of actions in the form of macros setting forth predefined actions, as depicted in block 302 in accordance with a preferred

embodiment of the present invention. The user then correlates or associates particular phrases to the actions, as illustrated in block 304. In other words, the user associates a voice phrase or ideal input event to a macro. The user then loads a template for a particular application, as depicted in block 306. Typically during this phase of the process, the voice recognition system may encounter background noise. This noise may match an entry within the template, i.e., meet a confidence factor of an entry within the set of phrases.

Detailed Description Text (10):
Alternatively, a command may be associated with the background noise phrase. For example, the command may disable the voice recognition system until some other event occurs. This approach allows for a dynamic training of the voice recognition system for background noise. The system may be trained to recognize different background noises, which decreases the probability that a background noise will be mistaken for a recognizable phrase within the set, i.e., the recognizable set now includes a confidence factor for the background noise.

Detailed Description Text (12):
A comparison of the digitized sound form to the digitized forms trained by the user within the template is performed by the voice recognition system detecting a sound or audio input event. Upon detecting background noises as defined by the interrupt criteria dynamically introduces sound phrases into the template. The symbol { } designate entries for phrases produced by the invention as can be seen in the PHRASE column in FIG. 4.

Detailed Description Text (13):
In accordance with a preferred embodiment of the present invention, an association of a null command to the entry for a created phrase may be made. The voice recognition system, upon detecting an audio input event, background or human voice, compares the sound to all entries within the template. A background sound also is referred to a "non-speech audio input event" and a human voice sound also is referred to as a "speech audio input event". A higher confidence factor exists for a comparison of a background noise to a background noise sample because the voice recognition system compares audio input events to a recognizable set of entries within the template.

Detailed Description Text (14):
Referring now to FIG. 5, a flow chart of a process for registering peripheral devices that can create noise or background sounds (non-speech audio input events) is depicted in accordance with a preferred embodiment of the present invention. The process begins by receiving user input in the form of names for the peripheral devices, as illustrated in block 502. The process then receives user input identifying interrupts for each of the peripheral devices, as depicted in block 504. Thereafter, user input is received by the process, designating associated communications ports for the peripheral devices, as illustrated in block 506. The process then receives user input as to the elapsed time for the recognition of devices, as depicted in block 508.

Detailed Description Text (17):
Referring now to FIG. 6, a flow chart of a process for training and updating a voice recognition system to detect and differentiate between sounds (also called "audio input events") that are speech audio input events or non-speech audio input events. The process begins by loading a device recognition table into active memory, as illustrated in block 600. The device recognition table is the data entered by the user and stored as illustrated in FIG. 5. The process then sets the interrupt vectors to intercept interrupts from the peripheral devices designated in the device recognition table before they reach the target application, as illustrated in block 602. The process then activates a monitoring service, as depicted in block 604. A monitoring service used to monitor for interrupts is well known to those of ordinary skill in the art and various methods may be employed in

accordance with a preferred embodiment of the present invention.

Detailed Description Text (19):
The process then awaits an audio recognition, as depicted in block 616. In other words, the process waits to see if a recognizable pattern, a pattern that meets a confidence threshold for an entry in the template, is detected. Upon the recognition of audio, the process then determines whether an audio interrupt has been received, as illustrated in block 618. An audio interrupt occurs when an input device, such as a microphone, detects an audio input event. If an audio interrupt has not been received, the process then determines whether the time has expired for recognition, as depicted in block 620, if time has expired for recognition, the process then clears the mark for the time that the interrupt is received, as illustrated in block 622, with the process then returning to block 606 to await an interrupt from a peripheral device. Referring again to block 620, if time has not expired for recognition, the process then returns to block 616 to await an audio recognition.

Detailed Description Text (20):
Referring again to block 618, if an audio interrupt is received, the process then proceeds to receive the audio pattern (the audio input event), as depicted in block 624. The process marks the time of reception of the audio pattern, as illustrated in block 626. Thereafter, the process determines whether the audio pattern is recognizable, as depicted in block 628. If the audio pattern is not recognizable, the process then proceeds to clear the mark for the time of reception of the audio pattern as illustrated in block 630. Thereafter, the process proceeds to block 622 as described above.

Detailed Description Text (21):
Referring again to block 628, if the audio pattern is recognizable, the process then subtracts the interrupt time for the peripheral device from the audio interrupt time to determine an elapsed period of time, as depicted in block 632. The process next determines whether the time period calculated is within an acceptable range, as depicted in block 634. If the time period is not within an acceptable range, the process proceeds to block 630 as described previously. On the other hand, if the period of time is within the acceptable range, the process then determines whether the user is to be notified of the recognition of the non-speech audio input event, as depicted in block 636. If the user is to be notified, the process then determines whether commands are to be executed, as illustrated in block 638.

Detailed Description Text (22):
Referring back to block 636, if the user is to be notified, the process then proceeds to notify the user according the recognition table definitions, as illustrated in block 640. Thereafter, the process also determines whether commands are to be executed, as depicted in block 638. If commands are to be executed, the process then executes the commands according to the recognition table, as depicted in block 642. Thereafter, the noise (non-speech audio input event) is stored as a recognizable template pattern, as illustrated in block 644. In other words, the non-speech audio input event is stored as an entry in the template. Thereafter, the process proceeds to block 630 as previously described. Referring again to block 638, if commands are not to be executed, the process proceeds directly to block 644. With reference now to FIG. 7, a flowchart of a process for analyzing audio input events in a data processing system is depicted in accordance with a preferred embodiment of the present invention. The process begins by identifying and recording a speech audio input event, as depicted in block 700. Next, the recorded speech audio input event is processed to create a first entry in a template, as illustrated in block 702. The process then identifies and records a non-speech audio input event, as depicted in block 704. The recorded non-speech audio input event is then processed to create a second entry in the template, as illustrated in block 706. Next, the process detects an interrupt, as depicted in block 708. Then,

an audio input event is detected after the detection of an interrupt, as illustrated in block 710. The process then identifies a non-speech audio input event by comparing the detected audio input event with the template, as depicted in block 712.

Detailed Description Text (23):
After the expiration of a period of time after the interrupt, the identified non-speech audio input event is processed to replace the second entry in the template for the processed non-speech audio input event, as illustrated in block 714. Additionally, in response to identifying a non-speech audio input event, a determination is made after the expiration of a period of time after the interrupt occurs as to whether a command is associated with the non-speech audio input event, as depicted in block 716. If a command is associated with the non-speech audio input event, the command is executed, as illustrated in block 718 with the process terminating thereafter. With reference again to block 716, if a command is not associated with the non-speech audio input event, the process also terminates. Blocks 714 and 710 both occur in response to an identification of a non-speech audio input event.

Detailed Description Text (25):
The present invention may be directed to intercept hardware interrupts or interrupts of the operating system. The registration service illustrated in FIG. 5 allows a user to specify the interrupts upon which recording should be activated for an audio input event. The user may adjust the sensitivity at which an interrupt should be interpreted as background noise or a non-speech audio input event. Predefined defaults may be set for existing devices, e.g., printers normally operate on interrupt 5H.

Detailed Description Text (26):
In accordance with a preferred embodiment of the present invention, a pre-process may be employed to evaluate if the audio input event detected should be compared to an existing null phrase or to create a new phrase. Such a process may involve the continuous employment of background noise to train the system for a particular noise phrase. In addition, null commands may be substituted for user supplied or system default commands. For example, the commands could issue a SAVE command for a word processor. With an increase in noise activity, such as interference, the user may desire to save the work presently completed. In such a situation, one of the background sounds matches an executable phrase.

Detailed Description Text (29):
In accordance with a preferred embodiment of the present invention, the fundamental problem of voice recognition systems involving differentiation of non-speech audio input events from speech audio input events is addressed. The present invention recognizes that peripheral devices may produce background noise and that a system may be allowed to essentially execute commands that are irrelevant to the applications in response to this background noise. The present invention provides a method and apparatus for allowing peripheral devices to affix digitized sound phrases within voice recognition sets, such as templates. The present invention provides further advantage over prior art methods in that the present invention does not need to always be activated. Once, a background noise is "trained" and registered into the template, the invention may be disabled or removed. This provides an advantage of freeing computer resources for other applications.

CLAIMS:

1. A method for analyzing audio input events in a data processing system, wherein said data processing system utilizes a template to analyze audio input events, wherein said data processing system includes a peripheral device that generates said audio input event and an interrupt, said method comprising the steps of:

identifying a speech audio input event;

recording said identified speech audio input event;

processing said recorded speech audio input event to create a first entry in a template;

identifying a selected non-speech audio input event which occurs in a selected environment;

recording said identified non-speech audio input event;

processing said recorded non-speech audio input event to create a second entry in said template; and

thereafter, distinguishing between a speech audio input event and a non-speech audio input event by comparing said audio input event to said template in response to detecting said interrupt and detecting said audio input event within a preselected amount of time wherein said non-speech audio input event is identified.

2. The method of claim 1, further comprising:

determining whether a command is associated with said non-speech audio input event in response to identification said non-speech audio input event; and

responsive to said command being associated with said non-speech audio input event, executing said command.

3. A method for analyzing audio input events in a data processing system, wherein said data processing system utilizes a template to analyze audio input events and wherein said data processing system includes a peripheral device that generates an audio input event and an interrupt, said method comprising the steps of:

identifying a speech audio input event;

recording said identified speech audio input event;

processing said recorded speech audio input event to create a first entry in a template;

identifying a selected non-speech audio input event which occurs in a selected environment;

recording said identified non-speech audio input event;

processing said recorded non-speech audio input event to create a second entry in said template for said processed non-speech audio input event;

detecting an interrupt;

detecting said audio input event, wherein said audio input event occurs after said interrupt;

identifying a non-speech audio input event by comparing an audio input event to said template;

responsive to identifying a non-speech audio input event occurring a preselected amount of time after said interrupt occurs, determining whether a command is associated with said non-speech audio input event; and

executing said command in response to said command being associated with said non-speech audio input event.

4. The method of claim 3 further comprising processing said identified non-speech audio input event occurring said preselected amount of time after said interrupt occurs to replace said second entry in said template for said processed non-speech audio input event.

5. An apparatus for analyzing audio input events, wherein said utilizes a template to analyze audio input events, wherein apparatus includes a peripheral device that generates an audio input event and an interrupt, said apparatus comprising:

first identification means for identifying a speech audio input event;

first recording means for recording said identified speech audio input event;

first processing means for processing said recorded speech audio input event to create a first entry in a template;

second identification means for identifying a selected non-speech audio input event which occurs in a selected environment;

second recording means for recording said identified non-speech audio input event;

second processing means for processing said recorded non-speech audio input event to create a second entry in said template for said processed non-speech audio input event; and

comparison means for distinguishing between a speech audio input event and a non-speech audio input event by comparing said audio input event to said template in response to detecting said interrupt and detecting said audio input event within a preselected amount of time, wherein said non-speech audio input events may be efficiently distinguished from speech audio input events.

6. The apparatus of claim 5, further comprising:

means for determining whether a command is associated with said non-speech audio input event in response to identification of said non-speech audio input event; and

responsive to said command being associated with said non-speech audio input event, means for executing said command.

7. An apparatus method for analyzing audio input events, where said apparatus utilizes a template to analyze audio input events and wherein said data processing system includes a peripheral device that generates an audio input event and an interrupt, said apparatus comprising:

first identification means for identifying a speech audio input event;

first recording means for recording said identified speech audio input event;

first processing means for processing said recorded speech audio input event to create a first entry in a template;

second identification means for identifying a selected non-speech audio input event which occurs in a selected environment;

second recording means for recording said identified non-speech audio input event;

second processing means for processing said recorded non-speech audio input event to create a second entry in said template for said processed non-speech audio input event;

first detection means for detecting an interrupt;

second detection means for detecting said audio input event, wherein said audio input event occurs after said interrupt;

third identification means for identifying a non-speech audio input event by comparing an audio input event to said template;

determination means, responsive to identifying a non-speech audio input event occurring a preselected amount of time after said interrupt occurs, for determining whether a command is associated with said non-speech audio input event; and

execution means for executing said command in response to said command being associated with said non-speech audio input event.

8. The apparatus of claim 7 further comprising means for processing said identified non-speech audio input event occurring said preselected amount of time after said interrupt occurs to replace said second entry in said template for said processed non-speech audio input event.

L46: Entry 7 of 21 File: USPT Oct 2, 2001

DOCUMENT-IDENTIFIER: US 6298370 B1
TITLE: Computer operating process allocating tasks between first and second processors at run time based upon current processor load

Drawing Description Text (37):
FIG. 32 is a further diagram of interrupt priority levels over time in process, device and system embodiments;

Detailed Description Text (19):
In some improved system embodiments, the OS controls multiple modular, stackable, concurrent computational resources (processors or hardware accelerators), and the improved system supports a wider variety of multimedia device emulation tasks. Modularity adds processing MIPs or elements, and the improved system gracefully orchestrates their operation with the host CPU/MMX for audio, video, graphics, communication and other functions. These modular and distributed processing elements in the improved system can better control latency for real-time events.

Detailed Description Text (66):
Time-slicing operating system code prevents an application from monopolizing the host by allotting runtime for the application in time slices thereby allowing other applications to be time-division-multiplexed. A preemptive multitasking OS further introduces a priority scheme to allow preemption of one task by another of a high priority. The improved USP system software granulates, or breaks up, applications into software objects called granules. Time-slicing and granulation do not conflict or introduce complications in each other's presence. Time-slicing and prioritization are ways used for scheduling in Windows. Time-slicing comes below prioritization in the scheduling scheme. A granule can simply be a software thread scheduled and run under the Windows regime.

Detailed Description Text (67):
A software decoder, for example, has lower priority than that of a hardware event. The VSP by means of hardware interrupts can naturally preempt a host-based program and work to advantage in the Windows OS scheduler environment. The VSP briefly interrupts the host to raise its priority with the Windows OS scheduler. If the host were to lock out interrupts, it would simply become a single-tasking system, therefore the host should not do so. Thus, VSP is a "very good citizen" for the Windows OS.

Detailed Description Text (123):
USP implements a software caching scheme to insert the VSP memory spaces into the host virtual memory space thereby utilizing the host's caching mechanism as well as its own for memory accesses. The program code and data for the VSP are continually cached into the DSP core or chip from the VSP wrapper program and data space in host (system) virtual memory for execution as shown in the VSP software caching model, FIG. 9 of incorporated U.S. patent application Ser. No. 08/823,257. Since the data processed by the VSP are real-time digital signals or non-cacheable data, a software (paging) caching scheme rather than a traditional Pentium CPU caching scheme is used for the VSP. A traditional L1, L2 type of write-back or write-through cache might have the undesirable effect of cache thrashing when used with non-cacheable data. The VSP software or paging cache acts as macrostore for DSPops executed in parallel with Host CPU/MMX instructions.

Detailed Description Text (138):
Deterministic response time for real-time applications is afforded when the VSP is used to guarantee processing time to the external events/interrupts and control latency due to its processing for the most critical (high-frequency portion) part of the real-time event processing. The VSP operations blend into the Windows OS operations for optimum execution. In real time systems, latency refers to the total time that it takes the host CPU to acknowledge and handle an interrupt. Consider a time interval occupied by high-frequency VSP interrupt handling followed by low-frequency host ISR and then non-time-critical Windows thread execution with a DPC. That time interval encompasses all operations that handle an external real-time multimedia interrupt, and can be substantially determined and controlled according to the processes of operation and architectural embodiments disclosed herein.

Detailed Description Text (142):
Advantageously, USP does not need an OS of its own. See FIG. 8 of incorporated U.S. patent application Ser. No. 08/823,257. Instead, USP uses Windows OS as its own OS via DirectDSP and the real-time VSP Kernel software (USP resource management is built into DirectDSP and the VSP kernel). This software architecture is both complementary and non-competing with the Windows OS. In the preemptive, multi-threaded, multi-tasking Windows OS, processes and threads are normally running at S/W IRQLs with lower priorities than the H/W IRQLs. Although threads can be raised to real-time high priority via software, they are still at or below IRQ2 (dispatch). In FIGS. 29, 30, 31 and 32, by tying a process or thread to a H/W event/interrupt (IRQ12-IRQ27), USP raises the process or thread priority to above other software (host-based) processes or threads.

Detailed Description Text (144):
Not only do VSPs efficiently handle real-time events and multimedia, they further enhance the Windows OS by virtualizing real-time Interrupts and DMAs. A VSP can even act as an MMX emulator/accelerator or a WDM accelerator accelerating the Windows OS.

Detailed Description Text (237):
The PCI bus master ISR and the PCI request queue provide system I/O functionality over the PCI bus. PCI I/O implies the transfer of data between any valid PCI address and ASIC RAM, external VSP RAM, or on chip RAM. PCI I/O, memory or configuration cycles can be performed over the PCI bus via the wrapper ASIC. Some PCI requests require VSP to copy from ASIC RAM to on chip RAM. A PCI request is posted in the PCI request queue to call for PCI I/O. If no other request is pending then the PCI bus master ISR is invoked immediately. If there are other requests pending then the request is placed in the queue according to its priority and eventually serviced by the PCI bus master ISR. When the PCI bus master ISR is invoked, it processes the next request in the queue. Each PCI request involves one or more PCI transactions. Where a single PCI transaction is required, the PCI bus master ISR commands the wrapper ASIC to perform the PCI transaction and return to the preempted code, see FIG. 34. This allows the currently executing task to continue execution while the PCI transaction is taking place. Also, other interrupts which occur can be serviced.

Detailed Description Text (484):
Assuming the StartIo routine finds that the transfer can be done by a single DMA operation, the StartIo routine calls IoAllocateAdapterChannel with the entry point of the driver's AdapterControl routine and the IRP. When the system DMA controller is available, an IRP next calls the AdapterControl routine DDAdapterControl to set up the transfer operation. The AdapterControl routine calls IoMapTransfer with a pointer to the buffer, described in the MDL at Irp.fwdarw.MdlAddress, to set up the system DMA controller. Then, the driver programs its device for the DMA operation and returns. When the device interrupts to indicate its transfer operation is complete, the driver's ISR DDInterruptService stops the device from generating interrupts and calls IoRequestDpc which executes another IRP to queue the driver's DpcForIsr routine DDDpcForisr to complete as much of the transfer operation as possible at a lower hardware priority (IRQL).

Detailed Description Text (498):
The DirectDSP HAL is 32-bit code residing at ring 0, the code having a combination

of C and assembly language for time critical functionality. The DirectDSP HAL conforms to Microsoft Windows conventions. Calls to the virtual machine manager (VMM) are made by the DirectDSP HAL to establish real-Lime priorities. The first call to the DirectDSP HAL causes the DirectDSP HAL to be loaded which in turn initializes the VSP hardware by allocating host memory resources for VSP program and data, filling these areas with each VSP load module (VSP code and data), configuring the VSP hardware, and boatloading the VSP. Next the VSP initializes the VSP kernel by retrieving code and data from host main memory. The DirectDSP HAL also locks down audio/modem application data in host main memory for bus master access by the VSP. Memory buffers of data are passed to the DirectDSP HAL for processing by the VSP. The DirectDSP HAL locks down the memory pages used by these buffers so that the Windows OS will not relocate the pages to disk. Once locked down the DirectDSP HAL acquires the physical memory addresses of the pages and stores them in memory for the VSP to use in accessing the data via the PCI bus. The DirectDSP HAL also passes data between the host and the VSP hardware via slave PCI accesses. The DirectDSP HAL communicates with the VSP via either data structures located in host main memory or in the VSP wrapper ASIC RAM. Interrupts from the VSP to host occur via the PCI Bus. Interrupts from the host to the VSP are generated by writing the appropriate register in the wrapper ASIC.

Detailed Description Text (599):
Memory arbitration according to a fixed priority system is performed on the non-DSP side of the DPRAM. The elements competing for the bus are:

Detailed Description Text (604):
The stereo codec is first priority since it operates at up to a 44.1 KHz sample rate, higher than the voice codec at second priority. Voice codec has data rate of 8 KHz, and ample buffers. The slave part of the PCI block has third priority because of the great need to avoid tying up the PCI bus, even though the slave is slower than the master. The PCI bus functions share fourth priority. Although PCI bandwidth is important to conserve, the PCI bus puts lots of data in place very quickly.

Detailed Description Text (1033):
In FIG. 27, interrelated improved processes 2700 related to operating system, DirectDSP HAL, and VSP Kernel have a multithreaded, multitasking OS 2710 with pre-emptive scheduler and realtime priorities and services 2720 coupled to API oval including DirectDSP improvement 1810 and DirectX 2510. Multi-threaded resource management 2600 services single VSP, or even multiple VSPs of FIGS. 122, 123, 126 for example. DirectDSP HAL 1830 couples to VSP kernels 1840.i for each VSP in the system.

Detailed Description Text (1036):
In FIG. 29, interrupt levels are utilized in connection with hardware interrupts and deferred procedure calls (DPCs) in process, device and system embodiments as shown. Notice use of both IRQs and DPCs. Priority raising occurs at arrow 2910 and 2920. Priority falls as indicated by arrows 2930, 2940 and 2950. This advantageous behavior used by the DirectDSP HAL and VSP Kernel is also depicted in FIGS. 30 and 32.

Detailed Description Text (1039):
In FIG. 32, interrupt priority levels over time in process, device and system embodiments are boosted by VSP hardware interrupt to host.

Detailed Description Text (1172):
If yes in step 11930, then decision step 11940 determines whether DSP MIPS are available. If yes in step 11940, operations go to step 11945 to allocate, load and run the new task on the DSP. If no in step 11940, the DSP is fully loaded (to maximum 100% test of step 11940, by contrast with step 11930 fractional level test) whereupon operations reach a decision step 11950. Step 11950 determines whether the current DSP task has higher priority (or no current DSP task has lower priority) compared to the new task which would be desirably run on DSP. A task running on the DSP is selected based on a priority table. If yes in step 11950 (no current DSP task has a lower priority), then the new task is run on the host at step 11935 whereupon step 11960 end of new task is reached. Notice that running the task on the host either impairs system performance by pushing the Host beyond optimal loading, or the

OS simply cannot run the new task on Host because Host goes over 100%.

Detailed Description Text (1173):
If no in step 11950, some current DSP task has a lower priority, and operations go to a step 11955 to shift current DSP task to the host or swap the current task out. "Swap" in step 11955 means the lower priority application is loaded for DSP but not currently executing thereon, and the new task will also be loaded on the DSP without shifting the current application to the host.

L12: Entry 11 of 30 File: USPT Jun 30, 1998

DOCUMENT-IDENTIFIER: US 5774701 A
TITLE: Microprocessor operating at high and low clok frequencies

Abstract Text (1):
A microprocessor incorporating a PLL circuit using a clock pulse having a relatively low frequency as an input clock signal of a reference frequency to form an oscillating pulse of a relatively high frequency by multiplying the input clock signal. In the microprocessor, the operation of the PLL circuit is stopped in the low-speed mode to supply the clock pulse of the relatively low frequency to the microprocessor as a system clock signal, and, in the high-speed mode, the PLL circuit is activated upon reception of an event requiring high-speed processing. Until the operation of the PLL circuit is stabilized and the request for high-speed processing comes, the above-mentioned clock pulse having the relatively low frequency is kept supplied continuously to the microprocessor as the system clock signal. This novel setup permits the high-speed switching of the microprocessor from the operating mode to the high-speed operating mode. Accordingly, the microprocessor may be kept operating until the output frequency of the PLL circuit is stabilized, thereby allowing the microprocessor to cope with an unpredictable situation such as the occurrence of a priority event or a failure.

Brief Summary Text (6):
In the microprocessor incorporating the PLL circuit, if the input clock signal and the output clock signal of the PLL circuit are selectively used as the system clock signal of the microprocessor, the operational flow of the microprocessor is as shown in FIG. 18 for example. When the microprocessor is in the low-speed mode (step ST11), the PLL circuit is in the stopped state, in which a clock signal having a relatively low frequency is supplied as the system clock signal to the remaining parts of the microprocessor including its central processing unit (CPU). This clock signal drives the microprocessor to operate at a relatively slow speed, thereby reducing the power consumption. If a high-speed event has accurred in step ST12 and a request for the high-speed processing for the event is recognized in step ST13, the PLL circuit is activated. In step ST15, it is checked whether the frequency of the output clock signal of the activated PLL circuit has been stabilized. Until the stabilization is confirmed, the system clock signal to the microprocessor is stopped in step ST14 to prevent unstable clock pulses from being supplied to the microprocessor. In step ST15, when the stabilization of the PLL circuit operation has been confirmed, the high-speed clock signal, or the output clock signal of the PLL circuit is supplied to the microprocessor as a system clock signal. This puts the CPU and other components of the microprocessor into the high-speed mode (step ST16), upon which the high-speed processing necessary for character recognition and the like starts. When completion of the processing such as character recognition and the like is recognized in step ST17, the microprocessor is put in the low-speed mode (step ST11).

Brief Summary Text (8):
However, the above-mentioned constitution thought out prior to the present invention has a problem. Namely, it takes a relatively long time to switch the system clock signal to the high-speed clock after a request for starting the high-speed processing is made. This is because a relatively long time is required for the stabilization of the PLL circuit operation. In addition, during this time, the supply of the system clock signal to the microprocessor is stopped, so that the operations of the remaining parts of the microprocessor including the CPU are

stopped. As a result, if a prioritized processing event or an unpredictable event such as a failure occurs during the pause of the system clock signal supply, no measure can be taken for it, thereby lowering the reliability of the microprocessor.

Brief Summary Text (9):
It is therefore a n object of the present invention to provide a microprocessor capable of quickly switching from low-speed mode to high-speed mode and taking measures against unpredictable events caused during the switching.

Brief Summary Text (11):
In carrying out the invention and according to one aspect thereof, there is provided a microprocessor incorporating a PLL circuit using a clock pulse having a relatively low frequency as an input clock signal of a reference frequency to form an oscillating pulse of a relatively high frequency by multiplying the input clock signal. In the microprocessor, the operation of the PLL circuit is stopped in the low-speed mode to supply the clock pulse of the relatively low frequency to the microprocessor as a system clock signal, and, in the high-speed mode, the PLL circuit is activated upon reception of an event requiring high-speed processing. Until the operation of the PLL circuit is stabilized and the request for high-speed processing comes, the above-mentioned clock pulse having the relatively low frequency is kept supplied continuously to the microprocessor as the system clock signal. When the operation of the PLL circuit is stabilized and the request comes, the oscillating pulse having the relatively high frequency formed by the PLL circuit is supplied to the microprocessor as the system clock.

Brief Summary Text (12):
According to the above-mentioned novel constitution, the microprocessor is switched from the low-speed mode to the high-speed mode at reception of the request for high-speed processing. In addition, after the PLL circuit is activated, the clock signal corresponding to the low-speed mode is kept supplied continuously to the microprocessor as the system clock signal, ensuring the uninterrupted operation of the microprocessor. This allows the microprocessor to take measures for the occurrence of a prioritized event or an unpredictable event such as some failure. As a result, the microprocessor is implemented that performs switching from the low-speed mode to the high-speed mode and takes measures against unpredictable events caused during the mode switching.

Detailed Description Text (9):
In the present embodiment, the operation of the PLL circuit is selectively halted according to an output signal of an OR gate OG1 supplied to a control pin PLLON of the PLL circuit. To one input pin and the other input pin of the OR gate OG1, a PLL control signal PLLON and a PLL standby signal PLLSB are respectively supplied from a clock controller CKC (to be described with reference to FIGS. 8 and 19), not shown. It should be noted that the PLL control signal PLLON and the PLL standby signal PLLSB are selectively made high when a predetermined register is set by the central processing unit to be described. Therefore, if the microprocessor is used on a portable information terminal for example, the PLL standby signal is made high at start of writing a character with the input pen, or at recognition of an event requiring high-speed processing to put the PLL circuit in the operating state in advance, thereby stabilizing the output frequency of the PLL circuit. Thus, the PLL circuit is ready for high-speed processing.

Detailed Description Text (32):
Next, in step ST2, if the occurrence of an event that requires the high-speed processing of the microprocessor is recognized (Yes), the high-level PLL standby signal PLLSB is outputted from the clock controller CKC in step ST3, thereby activating the PLL circuit. In step ST4, it is determined whether the frequency of the output signal of the PLL circuit has been stabilized. If the frequency has been found stabled, then, in step ST5, it is determined whether a high-speed processing start request has been made. It will be understood that, until the frequency of the PLL circuit output signal is stabilized, the PLL circuit remains in the low-speed operating mode (2), or in the activated state, and continues this state up to the point at which the high-speed processing start request is made. During such a state, the select control signal COSEL1 is kept low and the multiplexer MUX3 continues

outputting the system clock signal CK1 for low-speed processing formed from the clock pulse ck1. Therefore, the CPU and other components continue operating in the low-speed operating mode, but if an abnormal condition occurs on the system or a data processing request occurs to be processed before the high-speed processing to be performed, the request operation can be readily performed, thereby taking measures against unpredictable conditions.

Detailed Description Text (36):
The bus controller BSC is connected, at the other end thereof, with a peripheral bus P-BUS and an external bus E-BUS. The peripheral bus P-BUS is connected with peripheral unit controllers such as a refresh controller REFC, a direct memory access controller DMAC, a timer circuit TIM, a serial communication interface SCI, a digital/analog converter D/A, and an analog/digital converter A/D. The external bus E-BUS is connected with an external interface EXIF. The refresh controller REFC, the direct memory access controller DMAC, the timer circuit TIM, the serial communication interface SCI, the digital/analog converter D/A, and the analog/digital converter A/D are connected, at the other ends thereof, with an interrupt controller INTC. The interrupt controller INTC is connected with the central processing unit CPU via an interrupt request signal IRQ. The external interface EXIF is connected with a portable information terminal, an external memory or the like.

Detailed Description Text (37):
The interrupt controller INTC is further connected with a real-time clock circuit RTC. The real-time clock circuit RTC is supplied with a clock signal whose frequency is not changed by the control signal shown in FIG. 1. This allows the real-time clock circuit RTC to perform time management not affected by the control signal of FIG. 1. The real-time clock circuit RTC supplies an interrupt signal to the interrupt controller INTC at a predetermined time interval for example, thereby causing an interrupt request to the CPU at a predetermined time interval. Also, the interrupt controller INTC is supplied with an outside interrupt signal OINT. The outside interrupt signal OINT is transmitted to the CPU via the interrupt controller INTC.

Detailed Description Text (40):
The bus controller BSC controls the bus access made by the peripheral unit controllers connected to the peripheral bus P-BUS and, at the same time, controls the operations of these peripheral unit controllers. On the other hand, the refresh controller REFC, one of the peripheral unit controllers, controls a refresh operation of a dynamic RAM (Random Access Memory) provided as an external memory. The direct memory access controller DMAC supports high-speed data transfer between the external memory and the cache memory CACHE for example. The timer circuit TIM supports time management necessary for the central processing unit CPU. The serial communication interface SCI supports serial data transfer with an external communication controller or the like. The analog/digital converter A/D converts an analog signal supplied from an external sensor or the like into a digital signal of the predetermined number of bits. The digital/analog converter D/A converts a digital signal outputted from the CPU into a predetermined analog signal and outputs the resultant analog signal to the outside.

Detailed Description Text (41):
The interrupt controller INTC selectively receives interrupt requests from the peripheral unit controllers in a predetermined priority and transmits the received interrupt to the CPU as an interrupt request signal IRQ. The external interface EXIF controls and manages data transfer between the microprocessor MPU and an externally connected portable information terminal PDA and the external memory or the like, thereby providing interface between the MPU and these external devices. The bus controller BSC and the peripheral unit controllers are included in the second internal circuit that operates in synchronization with the system clock signal cks of relatively low frequency.

Detailed Description Text (43):
FIG. 10 shows one preferred embodiment of a clock supply path in the microprocessor MPU of FIG. 8. As shown in FIG. 10, the system clock signal CK1 of relatively high frequency outputted from the multiplexer MUX3 in the clock pulse generator CPG is

supplied to the central processing unit CPU, the multiplier MULT, the memory management unit MMU, and the cache memory CACHE included in the first internal circuit via the predetermined clock driver and then via clock switches CS1 through CS4. The clock switch CS1 is selectively put in the conductive state by the high level of a module enable signal CPEN. The clock switches CS2, CS3 and CS4 are selectively put in the conductive state by the high levels of module enable signals MUEN, TLEN, and CAEN respectively. Thus, the central processing unit CPU, the multiplier MULT, the memory management unit MMU, and the cache memory CACHE are selectively enabled by the high levels of the corresponding module enable signals. It should be noted that the clock switch CS1 put in the non-conductive state by the module enable signal CPEN is put in the conductive state again by an external reset signal or an external interrupt signal, not shown.

Detailed Description Text (44):
Next, the system clock signal cks of relatively low frequency outputted from the multiplexer MUX4 in the clock pulse generator CPG is supplied to the bus controller BSC, the refresh controller REFC, and the direct memory access controller DMAC of the second internal circuit via a predetermined clock driver and the via clock switches CS5 through CS7. The system clock signal cks is supplied to the timer circuit TIM, the interrupt controller INTC, the serial communication interface SCI, the digital/analog converter D/A, and the analog/digital converter A/D of the second internal circuit via another predetermined driver and then via clock switches CS8 through CS12. The clock switches CS5 through CS7 are selectively put in the transmitting state by the high levels of corresponding module enable signals BCEN, RCEN, and DMEN. The clock switches CS8 through CS12 are selectively put in the conductive state by the high levels of corresponding module enable signals TMEN, ICEN, UAEN, DAEN, and ADEN. As a result, the bus controller BSC, the refresh controller REFC, the direct memory access controller DMAC, the timer circuit TIM, the interrupt controller INTC, the serial communication interface SCI, the digital/analog converter D/A, and the analog/digital converter A/D are selectively enabled by the high levels of the corresponding module enable signals.

Detailed Description Text (47):
As described before, the system clock CK1 is formed according to the clock pulse ck1 of relatively low frequency when the microprocessor MPU is put in the low-speed operating mode and the select control signal COSEL1 is made low. When the microprocessor MPU is put in the high-speed operating mode and the select control signal COSEL1 is made high, the system clock signal CK1 is formed according to the clock pulse DV1 having a frequency four times as high. As a result, the speed of the operation of the first internal circuit including the central processing unit CPU is selectively increased or decreased to switch between the processing capabilities of the microprocessor MPU according to an event occurrence condition or the like.

Detailed Description Text (51):
The above-mentioned control register CS-Reg. has bits corresponding to the above-mentioned module enable signals. Namely, the CS-Reg. has a bit (Bcp) corresponding to the module enable signal CPEN, a bit (Bmu) corresponding to the MUEN, a bit (Btl) corresponding to TLEN, a bit (Bca) corresponding to CAEN, a bit (Bbc) corresponding to the BCEN, a bit (Brc) corresponding to RCEN, a bit (Bdm) corresponding to the DMEN, a bit (Btm) corresponding to the TMEN, a bit (Bic) corresponding to the ICEN, a bit (Bua) corresponding to the UAEN, a bit (Bda) corresponding to the DAEN, and a bit (Bad) corresponding to the ADEN, these bits not shown. Each of the bits is set with control data associated with the corresponding module enable signal. For example, the bit (Bmu) is set with control data associated with the module enable signal MUEN. The setting of the control data is achieved when the CPU specifies the control register CS-Reg. and writes the control data to the corresponding bit of the CS-Reg. via the P-BUS. For the central processing unit CPU to specify the control register CS-Reg., a predetermined address is assigned to the CS-Reg., but not necessarily limited thereto. The above-mentioned controller in the CKC forms each of the control signals according to the control data set to each of the bits of the control register CS-Reg. For example, When the CPU writes the control data "1" to the bit (Bmu) in the control register CS-Reg., the controller sets the module enable signal MUEN to the high level. When the CPU writes the control data "0" to the bit (Bmu), the controller sets the MUEN to the low level. Likewise, when the CPU writes the control data "1" (or "0") to the bits (Bcp),

(Btl), (Bca), (Bbc), (Brc), (Bdm), (Btm), (Bic), (Bua), (Bda), and (Bad), the controller sets the corresponding module enable signals CPEN, TLEN, CAEN, BCEN, RCEN, DMEN, TMEN, ICEN, UAEN, DAEN, and ADEN to the high level (or low level). Upon receiving the control signal from the interrupt controller INTC, the controller in the CKC forcibly resets each of the bits of the control register CS-Reg., but not necessarily limited thereto. By this reset, each of the bits of the CS-Reg. is set to the control data "1" that puts the above-mentioned clock switches in the conductive state.

Detailed Description Text (53):
Like the control register CPG-Reg., each of the bits of the control register CS-Reg. is also written (or set) with the control data by the program. Therefore, the supply of the system clock to each module is determined by the program. If an interrupt request or a reset request comes from the outside or an interrupt request comes from the real-time clock RTC, these requests are transmitted by the interrupt controller INTC to the controller in the CKC as the above-mentioned control signals. Therefore, if these requests come, the corresponding clock switches are put in the conductive state to supply the system clock signal to the corresponding modules. This setup restarts the supply of the system clock signal to the central processing unit CPU by the interrupt request (including the reset request) issued from the outside or the interrupt request issued from the real-time clock RTC even if the supply of the system clock signal to the CPU is in the stopped state.

Detailed Description Text (54):
FIGS. 11(A), 11(B), and 11(C) show schematic diagrams illustrating one preferred embodiment of an application that requires the processing speed switching in the microprocessor MPU of FIG. 8. In this embodiment, the microprocessor is for use in a portable information terminal PDA equipped with an input pen and controls the processing of the terminal. The portable information terminal PDA has an LCD (Liquid Crystal Display) covered with a pressure-sensitive sheet. The user writes characters on the LCD covered with the pressure-sensitive sheet by means of the input pen to enter data in the PDA. The input pen is provided with a micro switch, not shown, at its tip for identifying that the pressure-sensitive sheet is being pressed with the input pen. The open/close state of the micro switch is transmitted to the microprocessor MPU via a signal path, not shown. Receiving the open/close signal, the microprocessor MPU knows the input pen state, thereby recognizing the occurrence of an event that requires the high-speed processing and a request for starting the high-speed processing shown in FIG. 7.

Detailed Description Text (56):
Next, as shown in FIG. 11(B), when the user starts writing a character by pressing the input pen against the LCD cover with the pressure-sensitive sheet, the micro switch of the input pen is turned on, upon which the MPU recognizes the occurrence of an event that requires high-speed processing. Then, MPU instructs the clock controller CKC to make high the PLL standby signal PLLSB for the clock pulse generator CPG, putting the PLL circuit in the activated state. At this moment, the select control signal COSEL1 is left low as described before to let the first internal circuit including the CPU continue the low-speed operation. The instruction to the clock controller CKC in FIG. 11(B) is achieved by writing the control data "0" to the bits (Bon) and (B11) and the control data "1" to the bits (Bsb) and (Ben) of the control register CPG-Reg. That is, the instruction is achieved by making the central processing unit CPU execute the program that writes the control data. It will be apparent that the instruction may be executed only by the writing to those bits on which the control data is variable.

Detailed Description Text (59):
Each of the steps ST1, ST3, and ST6 in the operational flow of FIG. 7 is executed by the program that writes the control data to the control register CPG-Reg. as mentioned above. In step ST4, a time from activating the PLL circuit for example to the stabilization thereof is predetermined and this predetermined time is preset to the timer circuit TIM, the real-time clock circuit RTC, and the like. After the control data is written to the control register CPG-Reg. in step ST3, during the predetermined time set to the timer circuit TIM and the like, the program for step ST6 is adapted not to be executed even if the high-speed processing start request in step ST5 for example is made. This setup permits the execution of steps ST4 and ST5.

In the above-mentioned state ST7, the program for interpreting the entered character is executed. Upon completion of the execution, the processing goes back to step ST1. Namely, the writing of the control data to the control register CPG-Reg. is performed. Each of steps ST2 and ST5 may be achieved by a program such as making the central processing unit CPU detect the state of the above-mentioned micro switch via the external interface EXIF. Alternatively, an interrupt signal may be adapted to occur when the state of the micro switch changes and, according to this interrupt signal, the interrupt processing program as mentioned above for checking the micro switch state may be prepared to achieve each of steps ST2 and ST5. Thus, the operation flow of FIG. 7 may be achieved by the program to be executed by the CPU.

Detailed Description Text (62):
As shown in FIG. 12(B), in the microprocessor MPU according to the present invention, the PLL standby signal PLLSB is made high when a high-speed event, or an event that requires the high-speed processing is recognized, upon which the PLL circuit of the clock pulse generator CPG is activated. In the stabilization wait state until the output frequency of the PLL circuit is stabilized, the system clock signal CK1 is formed according to the clock pulse ck1 having the relatively low frequency as in the low-speed operating mode to made the first internal circuit including the CPU continue the low-speed processing.

Detailed Description Text (64):
Meanwhile, in the case of the microprocessor for executing the operational flow of FIG. 18, the PLL circuit of the clock pulse generator is activated upon recognition of a high-speed processing start request and the system clock signal is stopped until the output frequency of the PLL circuit is stabilized. Consequently, it takes a relatively long time from the recognition of the high-speed processing start request to the shifting of the microprocessor MPU to the high-speed mode. If, therefore, an event having a higher priority than the processing to be executed next or an emergency event such as a failure occurs while the system clock signal is in the halt state, there is no way for the microprocessor MPU to cope with such a situation, being put in system shut-down state, so to speak.

Detailed Description Text (65):
In the present embodiment, the PLL circuit is first activated at recognition of a high-speed event, the central processing unit CPU and other components are made to operate at the low speed until the output frequency of the PLL circuit is stabilized, and the CPU and other components are put in the high-speed mode upon recognition of the high-speed processing start request. This novel setup enhances the speed of the microprocessor mode switching operation and realizes the microprocessor being capable of coping with the occurrence of the events having higher priority or emergent events such a failure, enhancing the reliability of the microprocessor.

Detailed Description Text (66):
It will be apparent that, if the microprocessor according to the present invention is used on the above-mentioned portable information terminal PDA, the occurrence of an event required the high-speed processing may be recognized by the state of the input pen micro switch and, at the same time, the necessity for the character pattern recognition processing that requires the high-speed processing thereafter may be predicted. Therefore, at the time of recognizing the starting of a character entry, the PLL circuit of the clock pulse generator CPG may be activated to put the microprocessor in the standby state.

Detailed Description Text (87):
(1) In a microprocessor incorporating a PLL circuit using a clock pulse having a relatively low frequency as an input clock signal of a reference frequency to form an oscillating pulse of a relatively high frequency by multiplying the input clock signal, the operation of the PLL circuit is stopped in the low-speed mode to supply the clock pulse of the relatively low frequency to the microprocessor as a system clock signal, and, in the high-speed mode, the PLL circuit is activated upon reception of an event requiring high-speed processing. Until the operation of the PLL circuit is stabilized and the request for high-speed processing comes, the above-mentioned clock pulse having the relatively low frequency is kept supplied continuously to the microprocessor as the system clock signal. When the operation of

the PLL circuit is stabilized and the request comes, the oscillating pulse having the relatively high frequency formed by the PLL circuit is supplied to the microprocessor as the system clock. This novel setup permits the high-speed switching of the microprocessor from the low-speed operating mode to the high-speed operating mode.

Detailed Description Text (88):
(2) According to the setup of (1) above, the microprocessor may be kept operating until the output frequency of the PLL circuit is stabilized, thereby allowing the microprocessor to cope with an unpredictable situation such as the occurrence of a priority event or a failure.

Detailed Description Text (99):
Referring to FIG. 11, the method of recognizing the occurrence of a high-speed processing event and a high-speed processing start in the portable information terminal may be any other relevant methods. For example, the time at which character entry starts may be recognized as the occurrence of a high-speed processing event and the time at which a predetermined area on the LCD is specified with the input pen may be recognized as the request for the high-speed processing start. The external view of the portable information terminal PDA is not limited by the embodiment of FIG. 11. In FIGS. 12(A) and 12(B), the time relationship in each processing stage at the mode changing is not absolute. In FIG. 13, the constitution of the unit latch circuit ULT included in the input register IREG and others may take any other forms of embodiments. In FIGS. 14 and 16, the methods of controlling the output register OREG and the input register IREG are not limited to the ones illustrated and the timing relationships and valid levels thereof may take any other forms.

Detailed Description Text (101):
The typical advantages provided by the invention disclosed herein will be briefly described as follows. Namely, in a microprocessor incorporating a PLL circuit using a clock pulse having a relatively low frequency as an input clock signal of a reference frequency to form an oscillating pulse of a relatively high frequency by multiplying the input clock signal, the operation of the PLL circuit is stopped in the low-speed mode to supply the clock pulse of the relatively low frequency to the microprocessor as a system clock signal, and, in the high-speed mode, the PLL circuit is activated upon reception of an event requiring high-speed processing. Until the operation of the PLL circuit is stabilized and the request for high-speed processing comes, the above-mentioned clock pulse having the relatively low frequency is kept supplied continuously to the microprocessor as the system clock signal. When the operation of the PLL circuit is stabilized and the request comes, the oscillating pulse having the relatively high frequency formed by the PLL circuit is supplied to the microprocessor as the system clock. This novel setup permits the high-speed switching of the microprocessor from the low-speed operating mode to the high-speed operating mode. Accordingly, the microprocessor may be kept operating until the output frequency of the PLL circuit is stabilized, thereby allowing the microprocessor to cope with an unpredictable situation such as the occurrence of a priority event or a failure. As a result, the microprocessor is realized capable of switching from the low-speed operating mode to the high-speed operating mode may be performed at a high speed and coping with an unpredictable event caused at the mode switching.

CLAIMS:

18. A microprocessor according to claim 12, wherein the clock generator is activated by an occurrence of an event requiring a high-speed processing by the high-speed mode and the first internal circuit is put in the high-speed mode upon receiving a request for high-speed processing start occurring after the occurrence of the event.

19. A microprocessor according to claim 18, wherein the microprocessor is used in a portable information terminal having an input pen, recognizes a start of character entry by the input pen as the occurrence of the event requiring the high-speed processing by the high-speed mode, and recognizes an end of the character entry by the input pen as the request for the high-speed processing start.

27. In a microprocessor, a method of supplying clock signals to first and second modules of the microprocessor, comprising the steps of:

generating first and second clock signals;

supplying the first clock signal to the first modules and supplying the second clock signal to the second modules in a first mode of operation of the microprocessor;

determining a predictor event as a predictor for entering a second mode of operation of the microprocessor;

generating a third clock signal having a frequency higher than the first clock signal in response to determination of the predictor event, wherein the third clock signal is generated after a first time period, wherein the first clock signal is supplied to the first modules during the first time period;

determining a request to enter the second mode operation;

stopping the supply of the first clock signal to the first modules; and

supplying the third clock signal to the first modules, wherein the microprocessor operates in the second mode of operation.

31. The method of claim 27, wherein the predictor event comprises user input of data to a system incorporating the microprocessor.

32. The method of claim 27, wherein the predictor event comprises a beginning of pen input of data to the system.

L20: Entry 1 of 23 File: USPT Jan 14, 2003

DOCUMENT-IDENTIFIER: US 6507818 B1
TITLE: Dynamic prioritization of financial data by predetermined rules with audio output delivered according to priority value

Abstract Text (1):
A financial data system is disclosed that receives real-time data, uses a set of pre-determined rules to prioritize the data and provide a priority value, and then delivers the highest priority data by way of multiple audio channels. A key aspect of the invention is the use of data manipulation according to the priority value to adjust delivery volume, provide selective vocalization compression, add additional audio channels, or to override an existing comment when required. As a result of the invention, a significant amount of information may be aurally delivered to a user including properties of events as they change in response to changing financial conditions.

Brief Summary Text (6):
Thus, financial data systems are known that calculate and alert a client to technical indicators, time reminders, and price formations in visual or audio formats. However, each of these known systems has significant drawbacks. While information may be provided aurally, steps have not been taken to maximize the amount of data flow. While different indicators and messages may be provided, the indicators and alerts are only generally useful to make a user aware of isolated discontinuous events. They cannot convey information on the properties of the event such as its relative importance or magnitude in comparison to other events. Nor is there an ability to provide information on numerous events simultaneously.

Brief Summary Text (8):
The present invention is directed to a financial data system that receives real-time financial data using a data reception module and uses a pre-determined set of rules to normalize and prioritize the data, and to provide a priority value to the data. A first set of variables is provided in the data and a second set of variables is calculated using the first set of variables. The first and second sets of variables are then used to generate a listing of possible verbal comments ranked by the priority value. The possible verbal comments may be word or tone related.

Brief Summary Text (9):
Once prioritized, a sub-set of the data is broadcast using a plurality of audio channels. In a preferred embodiment, at least one audio channel plays background sounds while a second audio channel broadcasts specific comments ranked according to the priority value.

Brief Summary Text (11):
A key aspect of the invention is the manipulation of selected verbal comments before delivery using volume or vocalization compression corresponding to the priority value of each verbal comment. If necessary, while a current verbal comment is being delivered a higher priority new verbal comment may either override the current verbal comment or result in the addition of another audio channel.

Brief Summary Text (12):
In a preferred embodiment of the invention, the pre-determined rules include a decay value that affects the priority value of the possible verbal comments with each successive calculation of the second set of variables and the listing of the

possible verbal comments.

Brief Summary Text (13):
As a result of a combination of multiple audio channels, volume control, vocalization compression, and selective overriding, a tremendous amount of real-time financial information may be delivered to a user. Market events with greater significance and an accompanying louder sound or compressed delivery can be easily distinguished from events of lesser significance with lower volume and no vocalization compression. Mono-volume and non-compressed audio delivery systems are useful to make a user aware of isolated, discontinuous events. However, such systems can only alert a user that some event has occurred. Unlike the present invention, they cannot convey information on the properties of an event, such as its magnitude as part of a delivery system of numerous such events simultaneously.

Drawing Description Text (10):
FIG. 8 is a graph of a background sound volume assignment function with volume level (% of normal) as a function of the market comment's priority equation value.

Drawing Description Text (11):
FIG. 9 is a graph of a market comment continuous time-compression assignment function with delivery speed (% of normal) as a function of the market comment's priority equation value.

Detailed Description Text (6):
The flow of data 30 is described in greater detail in FIG. 2. First, a variable N, discussed in greater detail below, is initialized to 0. At a wait point 31, system 20 waits for either new financial data 30 or a pre-defined period of time since completion of a last comment 32 before passing control to a reception module 34. Typically, a pre-defined time period is chosen to reduce processing load. However, if it is desired for system to cycle continuously then the time period may be set to zero. Alternatively, if system 20 is only to run if new financial data 30 is received, then the time period may be set artificially high. In general, the purpose of having a pre-defined time period is to permit the controlled broadcast of lower priority comments even if no new financial data 30 is received.

Detailed Description Text (9):
The calculated variables associated with data 30 are also stored in arrays in database 28 by reception-module 34 along with previous values of the same variables. The variables are then normalized to values between 0 and 1 to bind the potential range of the values of the priorities of the comments.

Detailed Description Text (10):
Next, a determination is made at decision point 36 if a comment audio channel is available. If a comment audio channel is available, then data 30 is prioritized at point 38 using a logic engine or decision matrix. The logic engine sorts and sifts incoming data 30 based on a prioritization system derived from what is the most crucial and relevant data for market participants. For example, the logic engine uses the data inputs to analyze, sort, and prioritize price levels, changes in price levels, speed of change of price levels, trading volume, change in trading volume, flow of activity, timeframes, statistical, technical and other relevant numerical financial indicators and events to determine the appropriate audio comments, sequence of audio comments, and manner of delivery of the audio comments based on the generation of priority values corresponding to the data. To save on processor computational time, it is generally preferred that not all potential equations be solved unless certain criteria are met. Preferably, the maximum value of the priorities is approached asymptotically. The normalized values along with the most recent real-time variables (e.g., a-i above) are then used to calculate the priority value of each comment.

Detailed Description Text (12):
The priority value of each of the market comments is then defined by an equation involving mathematical combinations of pre-defined variables such as those listed above, as well as some additional special variables. One special variable is the "decay rate." A comment's decay rate helps determine how often a comment is reported. The priority value of a recently vocalized comment is decreased to reduce

the probability of a comment becoming too repetitious. The amount by which a recently vocalized comment's priority value is decayed is typically exponentially reduced over time although other types of decay rates may also be used. Typical comments are assigned a fast decay rate, and therefore, reported more often. Unusual comments that should be reported less frequently are assigned a slow decay rate. Sometimes comments are even assigned a negative decay rate, and actually increase in importance over time. Another special variable is the relationship of a comment to other comments. The comment relationship variable is an important factor for increasing or decreasing the priority of comments following the recently vocalized market comment(s).

Detailed Description Text (13):
The highest priority comment for a given audio channel is the comment with the highest resulting priority value. This is the comment that is aurally reported or vocalized as shown below.

Detailed Description Text (15):
The following examples are simplified to demonstrate a preferred method of generating market comments and assigning multiple audio output channels. To simplify this process, the number of potential market comments has been reduced to 1) Simple Bid Comment and 2) Simple Offer Comment. Further, the comment priority equations have been simplified to their core variables. Thus, for this example the simplified priority equations are:

Detailed Description Text (16):
Below is a list of Notations, Definitions, and Assumptions for use with the two equations: 1. N[ ]=normalization of data in brackets. Normalization can be achieved through various algorithms with the intent of bounding the values of input. 2. Normalization Equation. In the examples to follow the normalization equation will be: Variable/(Variable+3). 3. Bid Comment. A comment that states the price at which a potential buyer is willing to pay. 4. Offer Comment. A comment that states the price at which a potential seller is willing to sell. 5. Trade Comment. A comment that states the price at which something (stock, bond, financial or commodity product) has traded. 6. Size (or Volume). The quantity of something that has been bid for, offered, or traded. 7. Decay Value. As shown in the above Priority Equations, a comment's priority value is always reduced by its Decay Value. Whenever a comment has been vocalized its Decay Value is increased by a predetermined amount (the Decay Constant), reducing its priority to be said again. The "current" calculation cycle Decay Value is decayed by an algorithm (using a Decay Factor) and based on a time interval, or whenever a comment is selected for vocalization. The simplified equation used for calculating a comment's decay value for each new calculation cycle follows:

Detailed Description Text (34):
Once the priority values for all potential commentary equations have been calculated at point 38, at decision point 42 a determination is made if the highest priority comment of the analyzed time interval falls below a minimum threshold value. If it does then no comment will be made. System 20 will return to wait point 31 for new financial data 30 or a predefined time period since completion of the last comment 32.

Detailed Description Text (35):
If the priority value of the highest priority comment is greater then the minimum threshold, at decision point 44 a determination is made if pre-recorded audio files are to be used with optional data hooks (e.g., the insertion of numbers is following a pre-recorded statement) to broadcast specific variables associated with the data. Alternatively, at decision point 46 if data 30 is in text format, a speech synthesizer may speak the data. Otherwise, live streaming of the data is assumed at point 48. The use of pre-recorded audio files, speech synthesizers and live streaming are examples of data delivery means for converting data 30, as calculated, into verbal comments or tones.

Detailed Description Text (39):
The Priority Equation Value excludes the variables Decay Value and Previous Comment Relation. These variables are not used in this calculation because they are

important for guiding the sequence of comments, but not in Valuing a comment's absolute priority for the purpose of delivery.

Detailed Description Text (40):
As a result of the normalization process discussed earlier, the volume level equation for all comments must always be between 0 and 1 (0<=Value<=1). The normalization equations are designed so that average market data results in a priority equation value in a pre-determined value range.

Detailed Description Text (42):
The discontinuous model will set the volume output of market comments to a normal pre-determined volume level. Then, if a winning comment has a priority equation value above a specified threshold level, the volume level of this comment's vocalization will automatically be set to a higher level.

Detailed Description Text (43):
For example, the threshold level for higher volume vocalizations may be set to a priority equation value of 0.5 or greater. A Priority Equation Value of 0 to 0.49 would have a normal baseline volume while 0.5 to 1.0 could have a volume setting of 150% of the baseline volume. Using the data generated above, the first update's comment has a priority equation value of 0.4. Thus, it will be vocalized at a normal baseline volume level. FIG. 4 shows a graphical image of the wave pattern of a market comment reported at a normal baseline volume. The second and third market updates' comments each have priority equation values of 0.5, triggering the higher threshold volume setting. Therefore, the second and third market comments ("125 Bid" and "127 Offer") will be broadcast at a higher volume output level, 150% of normal. FIG. 5 shows a graphical image of the wave pattern of the same market comment shown in exhibit #1, but vocalized at 150% of normal baseline volume. FIG. 6 presents a direct graphical comparison of the two different volume levels. Of course, there also could be a series of threshold levels for multiple volume assignments.

Detailed Description Text (44):
The second basic way to automatically set volume levels for market comments is by continuous volume assignment. By way of continuous volume assignment, a unique volume level is set for each distinct priority equation value. Assuming that the baseline volume is set at 100%, every time a winning comment has a priority equation value of 0.5 the volume could be set to 150% of normal baseline volume. Every time a winning comment has a priority value of 0.3 the volume could be set to 103% of normal. Finally, every time a winning comment has a priority value of 0.2 the volume could be set to 91.3% of normal.

Detailed Description Text (45):
Essentially, volume levels are assigned to winning market comments as a continuous function. Using the same type of normalization equation used in the priority equations, each priority value has a unique volume assignment. Furthermore, the minimum and maximum volume levels can be pre-selected. For example, the minimum volume level could be 50% of normal and the maximum volume assignment could be 250% of normal. The volume assignment extremes are approached asymptotically as described by a graphical image of the normalization equation. Moreover, different types of market comments (Bid Family, Trade Family, etc.) can include different volume assignment normalization equations.

Detailed Description Text (46):
An example of using continuous volume assignment as applied to individual market comments is shown in FIG. 7. The volume level is on the Y-axis and the Priority Equation Value is on the X-axis. Winning comments with higher priority values will be assigned higher volume levels, as defined by the function graph. Volume levels are assigned in a continuous manner. Each specific priority value returns a unique volume level assignment.

Detailed Description Text (48):
Background sound is simply a way to help re-create the activity and feel of a real trading floor. For example, one or more separate audio channels may play or broadcast pre-recorded trading floor activity, which captures the yelling, screaming and trading of an exchange floor, or any other sound of interest in a continuous

loop. At the same time that data reception module reviews data 30 and determines if a comment audio channel is available at decision point 36, it also determines if one or more background sound channels are to be used as shown at decision point 52. If yes, then the audio priority values are calculated for the background sounds at point 38 as discussed above except that different pre-determined rules are applied for the background sounds as opposed to the comments, and the system continues as described. If no, then background sound is not used, terminating at point 54.

Detailed Description Text (52):
An example of how the Priority Equation Value for background volume may be calculated uses the same data as for the example above, but includes another variable called frequency, which is defined as the number of market updates per time interval. The table below shows the hypothetical values for the frequency variable immediately after the reception of a new market update.

Detailed Description Text (55):
Finally, there is another type of background sound. Such background sounds are typically played on a second or third background sound audio channel. The second type of background sound assigns abstract sounds, such as beeps, whistles, musical notes, or words such as "bought" or "sold" to various market activities. When a specific market activity (trade on offer side) occurs, the activity's assigned sound (e.g., "whistle" or "sold") is played. Controlling the volume of the sounds can convey the magnitude or importance of the specific market event. The volume level assignments are calculated in an identical manner to the method described for comment volume above. There are many market situations where a brief sound conveys maximum information in minimum time, especially with the volume control feature. This type of background sound provides a fast and efficient way to convey information to market participants.

Detailed Description Text (59):
As with volume control, sound compression may be either continuous or discontinuous. FIG. 9 shows an example of a continuous function that may be used for determining the delivery speed (time-compression) of an individual comment. The X-axis shows the comment's priority equation value (normalized and excluding the Decay Value and Last Comment Relation variables as discussed above because these variables are only useful in guiding the sequence of comments, but not in valuing a comment's absolute priority). The Y-axis shows the delivery speed assignment with values bounded from normal (100%) up to 70% faster than normal. For priority values below 0.25 the result is a slower than normal delivery speed. In practice, however, the slowest allowable delivery speed will typically be normal delivery speed.

Detailed Description Text (66):
In summary, the volume and data compression values are preferably determined as follows. Before actual vocalization, the highest priority comment's absolute equation value is compared with the equation value of other recent winning comments, and to a set of relative values. A volume level or data delivery rate for the highest priority comment is calculated from the comment's relative value. This volume level or data delivery rate assignment will be used to control the mode of the comment's vocalization. Relatively more important comments will be reported at a louder volume level and/or at a higher rate of data delivery. Less important comments will be reported at a lower volume level and/or a lower rate of data delivery. Although a comment may be the highest priority for the latest analysis interval it may not be an important comment in general. This method of volume and data delivery flow control allows for the "highlighting" of more important market information.

Detailed Description Text (67):
The use of volume control and data delivery time compression are important features of the invention. Moreover, they are examples of data manipulation means to alter an aural delivery of verbal comments according to priority values. Market events with greater significance and an accompanying louder sound or compressed audio can be easily distinguished from events of lesser significance and a quieter sound or uncompressed audio. Since traders are overwhelmed with information, most of which is visual, varying sound volume and data delivery compression provide a powerful way to focus a trader's attention on events of greater significance. Thus, a trader's

energy and concentration can be conserved for when it is needed in a busy market by being able to relax during slower periods and the accompanying low volume sound and uncompressed data flow. Without varying sound volume, traders have to either constantly watch the screen for market activity (which is exhausting to do all day long) or use a basic mono-volume system, which have limited utility.

Detailed Description Text (68):
Mono-volume alerts are useful to make traders aware of isolated, discontinuous events. However, mono-volume alerts can only alert traders that some event has occurred. They cannot convey information on the properties of the event, such as its relative importance or magnitude. In contrast, by varying the sound volume or compressing the delivery of an audible alert, the properties of an event can be conveyed to traders. This profoundly improves the usefulness of any audible alert.

Detailed Description Text (69):
For example, it is generally not useful and usually distracting to have a mono-volume alert beeping every time a common event occurs, such as buy orders being added to the current best bid. The only information being conveyed is that the event continues to occur. If the event is common and occurs often, this information is of little value to a trader and would be delivered at the expense of potentially distracting the trader. For this reason, mono-volume alerts are usually not used for commonly occurring events. Alternatively, an audible alert system in which an event's magnitude is algorithmically converted to a beep's volume or a compressed delivery of the beep can provide useful, continuous information, such as the frequency and size of the orders being added to the bid. This is valuable information to a trader, converting the potentially distracting beep into a meaningful continuous measure of a market's current state. Since the trader is receiving the information from the combination of audible alerts and changing volume levels he is free to visually concentrate on other areas of his trading screen. High concentrations of real-time data can be audibly conveyed to traders by carefully choosing the events to which sounds are assigned and properly designing their volume algorithms.

Detailed Description Text (73):
Specific applications of both volume and data delivery time compression include the following: Audibly conveying the buy and sell activity and the associated trade size through different tones or words such as "sold" or "bought" played at varied volumes or at different data compression rates. Playing a background sound of an actual trading pit which is varied in volume to reflect the amount of current trading activity. Low trading activity would be relatively quiet and fast market conditions would be loud. Playing either prerecorded verbal comments or text comments read by a synthesized speech program at varied volumes or at different delivery speeds to reflect the significance of the events being commented upon. The market depth (the entire book of bids including the bids below the current best bid, and all the offers including the offers above the current best offer) on the bid and offer can be represented with sound that could be algorithmically varied in volume to reflect the bullishness or bearishness of the market depth. The magnitude of bullish or bearish indicators could be represented by the varying volumes or delivery speed of their associated sounds. Orders being added to the bid side or offered side may be represented by tones or sounds and the volume of these tones in combination with their compression may be varied based on the size of the orders.

Detailed Description Text (77):
Once again the Priority Value excludes the variables Decay Value and Previous Comment Relation variables. As also noted above, the fourth update arrives while the third update's comment is being vocalized.

Detailed Description Text (81):
Once volume and vocalization compression are completed, background sounds play on background audio channels as shown at point 58. System 20 returns back to wait point 31 where module 34 waits for new financial data 30 or a predefined time period since completion of the last comment. Comments begin and end, freeing a comment audio channel as shown at point 60. As discussed in greater detail below, a variable N, used to control the number of times an over-ride of a current comment is possible, is set to 0 at the same time. Then, as also described below with respect to

over-riding current comments, a flag is checked at decision point 61 to determine if the data reception module should immediately activate without returning to wait point 31. If the flag is set then system 20 immediately returns to module 34. If the flag is not set, then system 20 returns to wait point 31. The above process is repeated until the market closes and the real-time streaming market data stops such that threshold 42 cannot be met or there is no additional data 30 to broadcast.

Detailed Description Text (82):
Priority Over-ride of Comments

Detailed Description Text (83):
As noted briefly above, the present invention includes the ability to institute a priority over-ride. This is yet another example of a data manipulation means. Occasionally there is new market data that is extremely important, so important that it needs to be reported immediately even before the current audio stream is completed, despite the use of one or more audio channels.

Detailed Description Text (84):
The use of the priority over-ride is only typically implemented if there are no comment audio channels available at decision point 36. If no such channel is available then at decision point 62 a determination is made if the data 30 has changed. If it has not changed, then system 20 returns to wait point 31. However, if it has changed then over-ride audio priority values are calculated at point 64. At point 64 it is possible to calculate all of the possible audio priority values as with point 38. However, it is more often preferred that only the priority values of a select few comments that are potentially allowed to override an existing comment be calculated to save on processing time. Once the priority values of one or more over-ride equations are generated, a determination is made at decision point 66 if the new comment is sufficiently more important than a current comment and an override threshold has been met. If it is not then system 20 sets a flag to "run data reception module" when a comment audio channel becomes available at point 68. When the flag is set to on, this is basically an indication that new market data 30 has been received while a comment was being verbalized. Thus, there is no need to return to wait point 31. Instead system 20 should return to data reception module 34. Then this portion of system 20 ends at point 70.

Detailed Description Text (85):
However, if the new comment is sufficiently more important than a current comment on a specific audio channel, then at decision point 72 the determination is first made if the current comment has already been overridden twice for this example, or by a pre-defined threshold number N. If the audio stream has already been overridden two times, then the current audio stream is not over-ridden and control passes to point 68 where system 20 sets a flag to "run data reception module" when a comment audio channel becomes above, as discussed above. Otherwise, at point 76 variable N is incremented by 1 and system 20 passes control to decision point 44. The idea is to avoid interrupting every market comment. Thus, there is a balance and trade-off between delivering the most relevant market data and making audio reports comprehensible. In a preferred embodiment, when a current comment is to be over-ridden and there are multiple comment audio channels available, the current comment on an audio channel with the lowest priority value is preferably over-ridden first.

Detailed Description Text (86):
An example of a priority over-ride situation follows. As with prior examples, ignore the Decay Value and the Previous Comment Relation variables in the priority equations. For purposes of the example, a priority threshold level of 0.2 or higher is set. Therefore, a new piece of market data must result in a priority value that is 0.2 or greater than the current comment's priority value. Using the same market data as above, a fourth market update is added. The fourth update arrives while the third update's comment is still being vocalized.

Detailed Description Text (87):
The third update's priority value is 0.5. Therefore, a new market update will need a priority value of 0.7 or greater to interrupt the "127 Offer" comment vocalization: Fourth Update: (Priority value calculation excluding decay and previous comment

relation) Bid Comment=N[129-122]=7/(7+3)=0.7 Offer Comment=N[127-130]=3/(3+3)=0.5

Detailed Description Text (88):
The bid comment wins with a priority value of 0.7. The priority value is 0.2 greater than the currently playing priority value. System 20 immediately stops the "127 Offer" vocalization in mid-stream. In its place the new Priority Over-ride comment of "129 Bid" is reported. Moreover, if volume control and compression control are implemented as discussed above and using the parameters in the corresponding examples above, "129 Bid" is vocalized with 150% greater volume level and 75% time-compression rate. As noted above, a fifth update could over-ride the fourth update, but if so then a sixth update would not be allowed to over-ride the fifth update through the use of the variable N.

Detailed Description Text (89):
When a comment is overridden by a priority comment the losing comment is not stored in a buffer or an output queue. Thus, there is no actual "buffering" in the classic sense. After a higher priority comment is vocalized a new calculation cycle begins. All of the comment priority equations are recalculated, including the equations for the comments that did not win the last calculation cycle or that were over-ridden. The previous cycle's priority calculations are discarded. Therefore, the over-ridden and losing comments will compete with all other comments for the right to be vocalized. However, there is a way to increase the priority of both over-ridden and losing comments into the next calculation round. In order to describe this process we must first define several types of comments.

Detailed Description Text (90):
There are two general categories of comments: "state" and "event" comments. Bid, Bid Size, Offer, and Offer Size are examples of "state" comments. As long as the bids, offers, and the like exist in a market then they will continue to be current and pertinent. Their "state" exists. If a "state" comment is over-ridden or loses in the previous calculation cycle it will compete as per normal with other comments for the next vocalization.

Detailed Description Text (91):
On the other hand, "event" comments are time sensitive. "Event" comments have three sub-categories: 1) Decaying value comments 2) Constant value comments 3) Increasing value comments

Detailed Description Text (92):
Trades and Trade Sizes are examples of decaying value "Event" comments. Trades are current the instant they occur and usually decay in importance over time. Therefore, if a trade occurs and is not vocalized (over-ridden or loses) in the current calculation cycle its priority to be vocalized is reduced or decayed as time passes. If a trade is vocalized it is flagged so that it will not be vocalized again.

Detailed Description Text (93):
A price handle change is an example of a constant value "event" comment. If the "big figure" of a market's price changes then the system needs to tell the listener what is the new big figure. The new big figure is referred to as a handle change. Some comments have a special variable called HandleChange in their priority equation. For example, if the bid has dropped from 105 to 98 the system should preferably announce "5 Bid" followed by "98 bid," as opposed to "8 bid." However, if a higher priority offer comment is vocalized instead of "98 bid" and the next bid is 95 then it is still necessary to say the big figure (handle), such as "95 bid" instead of "5 bid." Thus, system 20 preferably insures that the "ninety" part of the bid comment is eventually said by first setting the HandleChange variable equal to a constant, which increases the priority of the handle comment. The HandleChange variable remains equal to the constant (it is not decayed from one calculation cycle to the next) until the handle comment is finally vocalized. After vocalization the HandleChange variable is re-set to zero.

Detailed Description Text (94):
Finally, an example of an increasing value "event" comment is the announcement of an impending economic statistic. This type of comment, which is time dependent, will increase in priority while "waiting" to be vocalized. These types of comments have a

special variable called TimeSensitivity in their priority equations. For example let's assume that the monthly US unemployment figure is due out at 9:30 AM and our system should announce the following at 9:29 AM: "unemployment number at 9:30." Now suppose that frenetic market activity results in the generation of many high priority comments. The "unemployment" comment will increase in priority each calculation cycle as 9:30 approaches, ensuring that it will be vocalized well before 9:30 AM. In order to increase this comment's priority value, its TimeSensitivity variable will be increased after each calculation cycle in which this comment is not vocalized. When the comment is finally announced its TimeSensitivity variable will be re-set to zero.

Detailed Description Text (95):
In summary, an overridden comment is not buffered. The buffering effect comes through the priority equations and associated special variables, which as a whole act like an ecosystem. The system is self-correcting by keeping track of past activity through the manipulation of special variables (e.g., Decay Factor, Previous Comment Relation, HandleChange, and TimeSensitivity) that help to "guide" the vocalized output of system 20.

Detailed Description Paragraph Equation (3):
New Decay Value=(Previous Decay Value+Decay Constant (if applicable))*Decay Factor 8. Decay Constant. The Decay Constant is added to a comment's Decay Value after a comment has been vocalized. If a comment is not vocalized in the current calculation cycle, then the Decay Constant will not be added to a comment's Decay Value for the next calculation cycle. 9. Decay Factor. Usually less than 1 and greater than or equal to zero. The Decay Factor is multiplied by a comment's Decay Value after a specified time interval or after a comment has been selected for vocalization in order to exponentially reduce a comment's Decay Value. Thus, as the system moves forward in time through new comment cycles the ability of the Decay Value to suppress a comment is reduced. 10. Previous Comment Relation. As shown in the above priority equations, this variable is added to the priority equation value. When a comment is said it is often the case that another comment (or type of comment) should follow next. For example, if a bid comment is said then an offer comment should usually follow, unless newly received higher priority data takes precedence. For example, to increase the priority of an offer comment being said after a bid comment, the Previous Comment Relation variable in the offer priority equation is increased. In addition, some Previous Comment Relation variables will be influenced by comments made previous to the last comment, such as a sequence of comments. 11. Over-ride. The ability to stop a currently playing comment and play a higher priority comment. 12. Value Assignments for this example: When a Bid Comment is vocalized the Offer Comment's Previous Comment Relation Value=0.1; otherwise=0 When an Offer Comment is vocalized a Bid Comment's Previous Comment Relation Value=0.1; otherwise=0 Decay Constant=0.25 (for both bid and offer comment equations) Decay Factor 0.75 (for both bid and offer comment equations)

Detailed Description Paragraph Table (2):
Update # Winning Comment Priority Equation Value First Update "122 Bid" 0.4 Second Update "125 Bid" 0.5 Third Update "127 Offer" 0.5

Detailed Description Paragraph Table (6):
Priority Value Market Update # Bid Offer (1) Frequency Comment Market open: 120 bid 130 offer 1.0 First update: 122 bid 130 offer 0.4 1.5 "122 Bid" Second update: 125 bid 130 offer 0.5 2.1 "125 Bid" Third update: 122 bid 127 offer 0.5 3.0 "127 Offer" Fourth update: 129 bid 130 offer ? 3.6 ?

Detailed Description Paragraph Table (7):
Market Audio Priority Update # Bid Offer Value Freq. Comment Channel Market 120 bid 130 offer 1.0 open: First 122 bid 130 offer 0.4 1.5 "122 Bid" 1 update: Second 125 bid 130 offer 0.5 2.1 "125 Bid" 1 update: Third 122 bid 127 offer 0.5 3.0 "127 Offer" 1 update: Fourth 129 bid 130 offer 0.7 3.6 "129 Bid" 2 update:

Detailed Description Paragraph Table (8):
Priority Market Update # Bid Offer Value Freq. Comment Market open: 120 bid 130 offer 1.0 First update: 122 bid 130 offer 0.4 1.5 '122 Bid" Second update: 135 bid 130 offer 0.5 2.1 "125 Bid" Third update: 122 bid 127 offer 0.5 3.0 "127 Offer"

Fourth update: 129 bid 130 offer ? 3.6 ?

CLAIMS:

1. A financial data system comprising: real-time financial data, including a first set of variables and a second set of variables; a data reception module to receive said financial data; a set of pre-determined rules to dynamically prioritize said first set of variables and said second set of variables and provide a single real-time priority value to said financial data; at least one audio channel to broadcast said financial data; a data delivery module for converting said financial data into verbal comments; and a data manipulation module to dynamically alter an aural delivery of said verbal comments according to said priority value.

4. A financial data system as recited in claim 3, wherein said verbal comments are delivered at a volume greater than a base-line volume when said priority value is greater than a pre-determined threshold.

6. A financial data system as recited in claim 5, wherein said verbal comments are delivered in a compressed verbal format when said priority value is greater than a pre-determined threshold.

8. A financial data system as recited in claim 5, wherein said verbal comments are delivered at a volume greater than a base-line volume and in a compressed verbal format when said priority value is greater than a predetermined threshold.

9. A financial data system as recited in claim 1, wherein said data manipulation means comprises an override of a current verbal comment by a new verbal comment when said priority value of said new verbal comment is greater than a threshold value when compared to said priority value of said current verbal comment.

10. A financial data system as recited in claim 1, wherein a first set of variables are provided in said financial data and a second set of variables are calculated using said first set of variables, said first and second set of variables being used to generate a listing of possible verbal comments ranked by said priority value.

11. A financial data system as recited in claim 10, said pre-determined rules including a decay value that affects said priority value of said possible verbal comments with each successive calculation of said second set of variables and said listing of said possible verbal comments.

14. A financial data system as recited in claim 9, wherein at least a subset of said variables are normalized to generate said priority values between a pre-determined minimum and maximum value such that each of said possible verbal comments may be compared to all other possible verbal comments to determine a highest ranking priority comment for a specific audio channel.

16. A financial data system as recited in claim 1, wherein said data manipulation module comprises one of activating a new audio channel and overriding an existing verbal comment when said priority value of a new verbal comment is greater than a threshold value of existing verbal comment.

17. A financial data system as recited in claim 1, wherein said priority value of said verbal comments is adjusted to remove variables used in guiding a sequence of said verbal comments before using said data manipulation module.

21. A financial data system comprising: real-time financial data; a data reception module to receive said financial data, a set of pre-determined rules to dynamically prioritize said financial data and provide a real-time priority value to said financial data, a first set of variables provided in said financial data and a second set of variables calculated using said first set of variables, said first and second set of variables being used to generate a listing of possible verbal comments ranked by said real-time priority value; a plurality of audio channels to broadcast said financial data, a first audio channel broadcasting background sounds and a second audio channel broadcasting specific comments according to said real-time priority value; at least one of a pre-recorded audio file, voice synthesizer and

live audio streaming for converting said data into verbal comments; and at least one of a volume control, vocalization compression, a new audio channel and an overriding function to dynamically alter an aural delivery of said verbal comments according to said priority value.

22. A financial data system as recited in claim 21, wherein said pre-determined rules include a decay value that affects said priority value of said possible verbal comments with each successive calculation of said second set of variables and said listing of said possible verbal comments, said decay value being removed before using said data manipulation means and after selecting said verbal comments.

23. A financial data system comprising: real-time financial data, including a first set of variables and a second set of variables; a data reception module to receive said financial data; a set of pre-determined rules to dynamically prioritize said first set of variables and set second set of variables, and provide a single real-time priority value to said financial data; at least two audio channels being simultaneously used to broadcast said financial data; and a data delivery module for converting said financial data into verbal comments.

24. A method for vocalizing real-time financial data comprising the steps of: providing real-time data, including a first set of variables and a second set of variables; prioritizing said data according to pre-determined rules in a dynamic manner; generating a single priority value for said first set of variables and said second set of variables; converting said data into a plurality of possible verbal comments; selecting at least one verbal comment based on said priority value; choosing from one of a plurality of audio channels; manipulating said verbal comment and altering an aural delivery of said verbal comment according to said priority value; and delivering said verbal comment.

25. A method as recited in claim 24, including the further step of overriding a current verbal comment with a new verbal comment when said priority value of said new verbal comment is greater than a threshold value when compared to said priority value of said current value comment.

26. A method as recited in claim 24, wherein said manipulating step includes a step of adjusting automatically a delivery volume for said verbal comment according to said priority value.

27. A method as recited in claim 24, wherein said manipulating step includes a step of adjusting automatically a vocalization compression for said verbal comment according to said priority value.

29. A method as recited in claim 24, wherein said prioritizing step includes using a decay value that effects said priority value on a successive said prioritizing step for said possible verbal comments.

30. A method as recited in claim 24, wherein said converting step includes the sub-steps of normalizing said possible verbal comments between a pre-determined minimum and maximum value; and comparing each of said possible verbal comments to determine a highest ranking priority comment for a specific audio channel.

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
(17 AND 11).USPT.	14
(L17 AND L11).USPT.	14

Database: US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L18

Refine Search

Recall Text | Clear

Search History

DATE: Wednesday, May 28, 2003 Printable Copy Create Case

L20: Entry 14 of 23 File: USPT May 7, 1996

DOCUMENT-IDENTIFIER: US 5515372 A
TITLE: Method and apparatus for radio data control

Abstract Text (2):
Dynamic control of complex day-parting schedules minimizes data selection and coding errors and delays, provides priority access to the station's data channel and provides transparent scheduling of real-time data entries, while protecting against losses of real-time entries through overflow.

Brief Summary Text (25):
Time unit assignments in accordance with the present invention permit real-time data input, while preventing such things as pi and TA code bit errors. This dynamic scheduling feature also provides the flexibility necessary to coordinate diverse scheduling requirements, preventing timing errors and overflow losses.

Detailed Description Text (5):
The screen of the studio computer 13 displays impending on-air events, the station's schedule and the artist, title and catalogue number that is digitally recorded on each CD that is played, to guide the station operator, as is known in the art.

Detailed Description Text (6):
The RDS material broadcast by the transmitter 14 is assembled and encoded by RDS executable code resident in the RAM memory 20 of a dedicated RDS control computer workstation 22. Preferably, the RDS workstation 22 is one of the well-known DOS/IBM compatible computers having an Intel i286.TM. model processor and MS-DOS version 3.3, or higher. During the broadcast day, the RDS routines and files are in RAM memory 20 that is operated as a virtual "RAM disk" drive. For coordination with transmitter logging and control, the RDS workstation should have an Intel i486.TM. model processor and MS-DOS version 6.0, or higher. The RDS workstation 22 coordinates stored and real-time RDS data, and provides scheduling information and control options at the studio 10 through the studio automation computer 13.

Detailed Description Text (11):
Real-time Commands and Data

Detailed Description Text (12):
Commands and data can also be entered in real-time from the RDS workstation 22, or from remote sources. For example, the CD player 36 that reads digital CD artist, title and catalogue number data, the keyboard of the studio computer 13 or the TA switch 35 on the air console 11 in the studio 10, may provide data or commands at any time over the RS-232 cable or modem link. All real-time data entries from the studio 10 and any outside sources 37 are stored in respective data buffers 39 that are monitored by the dynamic sequencer 28.

Detailed Description Text (13):
For example, if the station leases RDS paging service channels or a transparent data service channel, the RDS workstation will have respective buffer memory areas 39 for receiving the real-time pager and transparent data entries. This real-time data is transferred from the respective buffer 39 to locations in the FILES 41, 42, 44 and the CURRENT TABLE 26 by the editor routine 40, under the control of the dynamic sequencer routine 28.

Detailed Description Text (20):
The flexible real-time data flow and automated parallel multiformat data group generation provided by dynamic sequencing in accordance with the present invention are explained further below.

Detailed Description Text (23):
The COMMAND TABLE 30 only lists reassignments, however, scheduled changes to the CURRENT TABLE. The OUTPUT TABLE defines the complete RDS schedule of static assignments, which also include TYPE 3 location data, TYPES 1, 4 and 7 for radio paging and TYPE 5 in-house transparent data. Blocks of TYPE 9 emergency data groups may also be inserted into the OUTPUT TABLE at any time in response to real-time data inputs.

Detailed Description Text (24):
The combination of RDS real-time display data entries and paging and transparent data entries, as well as traffic announcement switching and emergency data, imposes a substantial additional scheduling burden on the broadcaster. RDS standards require immediate insertion of the traffic announcement TA switch bit signal, and of emergency data code groups. RDS paging channel control requires a specific, second-by-second, RDS broadcast schedule. Furthermore, the real-time data is subject to overflow unless dynamically allocated within the RDS broadcast schedule.

Detailed Description Text (27):
Suitably-equipped receivers use TYPE 9 data for transmitting channel-addressed data to emergency services teams or to select sampled-voice sequences providing auditory messages that override all other audio material. In the event of an emergency, TYPE 9 groups must supersede all other scheduled RDS services, but in an orderly fashion so that the superseded real-time material is delayed rather than lost. Dynamic sequencing permits orderly reassignment of real-time data superseded by this TYPE 9 emergency service, while continuing to update static assignments to the CURRENT TABLE 26 in accordance with the COMMAND TABLE 30.

Detailed Description Text (34):
Radiotext messages are entered into the RADIOTEXT FILES 42, without reference to which Group format will be broadcast. Code-group format and sequence is determined on-the-fly for these entries whether they are to be broadcast immediately, as real-time data, or stored in the RADIOTEXT FILES 42. The difference between the RDS formats providing 64 characters 3 times a second in TYPE 2A groups, but 32 characters 6 times a second in TYPE 2B groups, for instance, is transparent to the operator. The operator merely enters radiotext and its duration value. The parallel generation of Long-PS codes with the Radiotext codes within the same 2-second time unit is transparent to the operator who enters the radiotext.

Detailed Description Text (43):
Dynamic, statistical sequencing of the RDS TYPES broadcast protects parallel group generation from disruption by the delays encountered in subdividing Long-PS text. Because the desired frequency with which groups are broadcast each minute is predetermined, rather than their actual sequence, the sequencer routine maintains order in the data flow, despite variable data volume, competing data priorities and delays.

Detailed Description Text (46):
When data levels in one of the real-time data buffers 39 reaches a given highwater mark, indicating that the real-time data could overflow its buffer 39, the dynamic sequencer 28 changes into its overflow mode.

Detailed Description Text (47):
In the overflow mode of operation, the dynamic sequencer 28 causes the editor routine 40 to use an MIX LIMIT TABLE stored in the STATIC FILE 42 in building its OUTPUT TABLE, rather than the NORMAL MIX TABLE. This MIX LIMIT TABLE defines the station's rock-bottom, baseline RDS broadcast needs and commitments, permitting accelerated data transmission from a full real-time data buffer 39:

Detailed Description Text (49):
If the data level remains above the high water mark for longer than a predetermined

TOLERANCE period, the RDS workstation forces a halt at the source of the data transmission causing the existing highwater condition, to prevent loss of data transmitted to the station. In the preferred embodiment this is done through the handshake protocols of the packet data communications methods used by the RDS workstation 22 to acquire real-time data.

CLAIMS:

2. The apparatus of claim 1 further comprising:

a real-time data input; and

a stored-data data input, said editor cooperating with said data inputs to supply real-time data to predetermined types of data segments, and said sequencer supersedes the data for a display assigned at said respective predetermined times in accordance with said predetermined display assignment list with a display from said real-time data input, whereby real-time display data is dynamically scheduled for broadcast.

3. The apparatus of claim 1 further comprising an input buffer for storing real-time data, and a limit table, said mix table being superseded by said limit table in the event of a highwater condition in said real-time input buffer so that more data segments to which real-time data is supplied are assembled into said information, said limit table listing minimum frequencies for respective data segment types, whereby real-time data is selected for broadcast by dynamic assignment and the risk of overflow is reduced.

8. The apparatus of claim 2 further comprising a buffer overflow limiter having a predetermined overflow tolerance time, said buffer overflow limiter interrupting said real-time data when a buffer overflow tolerance time has expired.

11. The method of claim 10 further comprising the sheds of:

supplying real-time data to predetermined types of data segments; and

superseding the data for a display assigned at said predetermined times in accordance with said predetermined display assignment list with real-time data, whereby real-time display data is dynamically scheduled for broadcast.

12. The method of claim 10 wherein the display assignment list and a real-time input buffer provide data and displays for said data segments, said method further including the step of:

superseding the mix table with a limit table in the event of a highwater condition in said real-time input buffer so that more data segments in which real-time data is supplied are assembled into said information, said limit table listing minimum frequencies for respective data segment types, whereby real-time data is selected for broadcast by dynamic assignment and the risk of overflow is reduced.

17. The method of claim 11 further comprising the step of interrupting the supply of real-time data when a buffer overflow tolerance time has expired.

L20: Entry 19 of 23 File: USPT Jan 4, 1994

DOCUMENT-IDENTIFIER: US 5276295 A
TITLE: Predictor elevator for traffic during peak conditions

Abstract Text (1):
A computer controlled elevator system (FIG. 1) including signal processing means for dynamically computing the population spread or density of the buildings, i.e., the number of elevator users in a building on a floor-by-floor basis, including the lobby, and to use such information to compensate for traffic shifts occuring in connection with the up-peak period in which dynamic channeling is used for an elevator car assignment scheme based on prediction methodology, all in accordance with an algorithm (FIG. 3). If, for example, the prediction methodology predicts that the up-peak dynamic channeling scheme should begin but the real time data has not detected any beginnings of an up-peak traffic pattern, the prediction methodology is over-ridden until the real time data finally picks up such a pattern. Additionally, if the floor population spread which is derived from real time data indicates that one or more floors individually have received all of their expected floor population, those floors are devalued to a nominal "priority" basis of "1" in the dynamic channeling scheme, even though the prediction methodology predicts the arrival of additional people for those floor(s) in the remaining up-peak time set by the system. Thus, "too early start" (FIG. 2A) and "too late end" (FIG. 2B) of dynamic channeling are avoided.

Brief Summary Text (12):
The present invention relates to elevator systems and more particularly to computer controlled systems which use predictions of future traffic conditions based on past or historic data, as well as real time events, as a guide to, for example, assigning elevator cars to certain floors for "channeling" for the operation of the system during up-peak periods. More particularly, the present invention relates to the timing of when and how the historic and real time data are combined for making the predictions and even more particularly to techniques for compensating for significant traffic shifts which impact on a peak period, with the up-peak period being the particularly preferred application of the invention.

Brief Summary Text (15):
Dynamic channeling provides a way of balancing the building traffic density evenly among the elevator cars in a building. In channeling generally, the floors above the main floor or lobby are grouped into sectors, with each sector consisting of a set of contiguous floors and with each sector assigned to a car, with such an approach being used during up-peak conditions. For dynamic channeling, rather than merely assigning an equal number of floors to each sector, prediction methodology is used for estimating the future traffic flow levels for the various floors every short time interval, for example, every five (5) minutes based on past events or traffic conditions. These traffic predictors are then used to more intelligently and dynamically assign the floors to more appropriately configured sectors, having possibly varying numbers of floors to optimize the effects of up-peak channeling.

Brief Summary Text (16):
Thus, a modern day, computerized elevator system for an office building continuously monitors and records elevator-related, significant events occurring in the building, preferably for every minute or short interval of the day, at least during the normal business day, and every day of the year, at least for every business day. Based on the data resulting from the building's elevator usage, a series of predictions are

performed to estimate the traffic density during the next few upcoming intervals, each of which intervals usually is a relatively short period of time, typically of the order of some few minutes, e.g., as noted above, five (5) minutes.

Brief Summary Text (19):
On the other hand, real time prediction is a prediction based on much more recent data collected over a sufficiently short period of time, usually involving some minutes, to effectively be considered "real time" for the time period for which the prediction is being made. It thus predicts traffic based on the events or data of only the past some minutes, rather than the past few days.

Brief Summary Text (20):
Depending on the number of intervals being "looked ahead" and the type of prediction(s) involved, typically a real time prediction uses a number (one or more) of the past intervals prior to the current interval. For example, at 9:15 AM, the real time prediction might use the data collected during the last three, five (5) minute intervals of, e.g., 9:00 AM to 9:05 AM, 9:05 AM to 9:10 AM, and 9:10 AM to 9:15 AM. Based on these three sets of collected data, the real time prediction predicts the expected traffic for the next five (5) minute interval in a way that matches or at least approximates the current traffic arrival curve.

Brief Summary Text (27):
Thus, if there are any such abnormal or unusual shifts in the traffic pattern from the historic pattern(s) in either direction, i.e., early or late arrival of the passengers from the predicted conditions or events, the prior standard methodology could cause on these some few "abnormal" days the initiation of up-peak channeling at a time not in sync with the actual traffic pattern and/or maintain such up-peak channeling beyond the need for such channeling.

Brief Summary Text (33):
However, the relative values for "a" and "b" preferably were determined as follows. When the up-peak period started, the initial final predictions preferably assumed that a=b=0.5, namely the factors were at least initially to be treated equally. Further predictions were then made at the end of each minute, using the past several minutes data for the real time prediction, as well as using the historic prediction data.

Detailed Description Text (4):
One application for the present invention is in an elevator control system employing microprocessor-based group and car controllers using signal processing means, which through generated signals communicates with the cars of the elevator system to determine the conditions of the cars and responds to, for example, hall calls registered at a plurality of landings in the building serviced by the cars under the control of the group and car controllers, to provide, for example, assignments of the hall calls to the cars, or, during up-peak conditions assigning the cars to various floor sectors using, for example, dynamic channeling in which the assignment is based at least in part on combined prediction values including real time and historic data. An exemplary elevator system with an exemplary group controller and associated car controllers (in block diagram form) are illustrated in FIGS. 1 and 2, respectively, of the '381 patent and described in detail therein, as well as in some of the related applications referred to above.

Detailed Description Text (29):
Since real data is used for the current floor density, the prediction may be discounted. This is done by inactivating the effect of that floor in the dynamic channel creation process. In other words, instead of giving that a higher priority due to the expected or predicted twelve people, it is given a normal "priority" or status for only one (1) person arriving. Thus this floor, like all the other floors, will receive service, but it will be regular service and not high priority service. This allows greater emphasis or higher priority service to be given to the other floors in which the pre-determined floor populations have not yet been satisfied.

Detailed Description Text (35):
Since population density or spread data based on real time data is used in the invention, some understanding and discussion of this aspect of the elevator system

is desirable for the complete understanding of the present invention. However, it should be understood that the methodology for pre-determining a building's floor population spread is not directly part of the present invention, and any appropriate methodology, particularly one that uses real time data such, as for example, de-boarding and boarding counts, can be used in this respect.

L20: Entry 5 of 23 File: USPT Aug 15, 2000

DOCUMENT-IDENTIFIER: US 6105048 A
TITLE: Apparatus and method for the real-time processing of a plurality of tasks

Abstract Text (1):
An apparatus and a method for the processing of a plurality of tasks by a processor of a real-time data processing installation, in which each task is dynamically allocated a priority according to its urgency, after which the tasks are processed by a processor. For this purpose, for each task to be processed, the generation of an initiation cell having a number of t- and i-bits is provided, as is the entering of this initiation cell into a free cell of a contention unit. The time sequence information contained in the t-bits of all cells of the contention unit are compared with one another in order to determine the task containing the smallest time sequence information. In case the determined task does not agree with the previously determined task, an interrupt signal is sent to the processor in order to introduce the processing of this newly determined task.

Brief Summary Text (2):
The present invention relates to an apparatus and a method for the processing of a plurality of tasks in a processor of a real-time data processing installation in which each task is to be processed individually by the processor according to a dynamically allocated priority and within a predetermined time period.

Brief Summary Text (3):
In a real-time system of the type named above having a dynamic priority allocation, the priority of a task or of an interrupt is assigned according to the relative urgency, in relation to the time at which a task is available for processing or an interrupt is requested.

Brief Summary Text (4):
In many of the previously used systems, a static priority allocation is used for tasks and interrupt requests that does not change over the course of time. In order to enable the timely processing of all tasks in such a system with n independent periodic tasks, the utilization of processor capacity may not exceed the theoretical value of n(2.sup.1/n -1). For large n, this value converges rapidly to ln2=0.69. That is, the utilization of processor capacity in such a system can be a maximum of only 70%, and the processor requires at least 30% idle time in order to be able to process all tasks in a timely fashion.

Brief Summary Text (5):
For the improvement of the utilization of the processor capacity, and thus of the performance of the data processing apparatus as a whole, the realization of a dynamic priority allocation has already been tried. A theoretical foundation for such a dynamic priority allocation, with which a processor capacity utilization of 100% can be achieved, can be found in the article entitled "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," in: Journal of the Association for Computing Machinery, vol. 20, January 1973, pp. 46-61. An algorithm for a priority allocation designated Earliest Deadline First (EDF) is therein described. According to EDF, of all the tasks awaiting processing, the one for which the time period for its processing expires the earliest receives the highest priority. In the article named, among other things there is also a comparison of the theoretical performance of a system having a static priority allocation with one having a dynamic priority allocation, and also a comparison of these systems with a

mixed priority allocation.

Brief Summary Text (6):
Although with a dynamic priority allocation according to EDF a utilization of processor capacity of 100% can be achieved, the performance of the system is not improved, since the processor must additionally process the algorithm for the priority allocation. In practice, it has turned out that in some cases the algorithm for the priority allocation causes an overhead of up to 80% of the processor time, so that only 20% of the processor time is available for the actual processing of the tasks.

Brief Summary Text (7):
A reduction of the utilization of processor capacity by the processing of the priority allocation algorithm can be prevented by providing an additional processor for processing the priority allocation algorithm, the additional processor being exclusively responsible for this task. A solution variation of this type can be found in the article "Implementation and Evaluation of a Time-Driven Scheduling Processor," in: EEE Transactions on Computers, 7/88, p. 172ff.

Brief Summary Text (10):
It is an object of the present invention to create a method and an apparatus in which the utilization of processor capacities with the use of a dynamic priority allocation according to EDF, in an apparatus without an additional processor, can be substantially increased in relation to known systems. Such a solution should also be able to be realized without high expense.

Brief Summary Text (11):
The above task is solved, according to the invention, by means of a method for processing a plurality of tasks by a processor of a data processing apparatus, in which each task is to be processed according to a dynamically allocated priority individually and within a predetermined time period. The method comprises the following steps:

Brief Summary Text (14):
Likewise, the posed task is solved by means of an apparatus for the processing of a plurality of tasks by a processor within a predetermined time period according to a dynamically allocated priority, in which apparatus, for an external task to be processed, means are provided for the generation of an initiation cell, consisting of at least a plurality of t-bits, which contain a time sequence information, and of a plurality of i-bits, which contain an index information. The apparatus additionally comprises means for entering the initiation cell of each task to be processed into a contention unit (PCU), in which a plurality of contention cells is provided for this purpose, and also comprises means for the comparison of the time information contained in the t-bits of all cells of the contention unit, in order to determine the task that contains the earliest expiring time information. Means are also provided for the transmission of an interrupt signal to the processor, in order to introduce the processing of the newly determined task, in case the determined task does not agree with the previously determined task just being processed by the processor. The time sequence information contained in the t-bits indicates, in the standard way, the time within which the processing of the task has to occur, while the index information contained in the i-bits refers to the task control block of the data processing apparatus.

Brief Summary Text (16):
improve the utilization of the capacity of the processor, since the processor does not have to process any overhead code for the priority allocation. The processor can thus be loaded with tasks up to the theoretical limit of 100%. The hardware expense for the contention unit is extremely low in comparison with an additional processor. A contention unit can, for example, comprise a simple static RAM memory block or simple binary counters, in which the contention cells are stored.

Brief Summary Text (17):
In an embodiment of the method or of the apparatus, the initiation cell is advantageously generated by means of a process initiation unit, in which, for each admitted external event that triggers a task for processing by the processor, an

initiation cell with t- and i-bits that are pre-coded or initiated by the processor is contained. It is further provided within the scope of the invention that for an internal task to be processed, e.g. a subtask, the t- and i-bits are generated for a contention cell by the processor and are entered into the contention unit.

Brief Summary Text (19):
In addition, it has proven advantageous if each cell of the contention unit additionally contains f-bits that are provided for an intertask communication, e.g. semaphore, event flags or message boxes, by means of which the participation of a contention cell in the comparison of the time sequence information contained in the t-bits can be temporarily suppressed. If a task must await a particular precondition, an f-bit is set in the contention cell of this task, in order to exclude it temporarily from the comparison of the time sequence information contained in the t-bits. If the preconditions for the further processing of these tasks are fulfilled, the f-bit of the allocated contention cell is reset, so that this cell can subsequently take part again in the comparison of the t-bits. In this way, an active and an inactive state of a contention cell can be represented in a simple way.

Detailed Description Text (3):
A plurality of signal lines are connected to the process initiation unit PIU, which respectively represent one external event by means of which a particular task can be signaled for processing by the processor CPU (here not shown). For this purpose, the process initiation unit comprises for each external event signal respectively one initiation cell IC (Initiation Cell), which are all combined into an array ICA (Initiation Cell Array). Each initiation cell IC contains: a determinate number of bits, namely a number n of t-bits, which indicate a relative time information for the corresponding task, within which time interval the task is to be processed at the latest by the processor CPU in order to satisfy the requirements of the real-time processing; a plurality of i-bits, which form an index that refers to the task control block TCB of the data processing apparatus, where further information for the processing of the respective task is stored; and a plurality of f-bits, which are used as flags, e.g. for inter-task communication. The t-bits can already be pre-coded for each possible task in the array ICA or can be written in the array ICA at the startup of the data processing equipment. The i-bits are written in the allocated cell in the initiation phase of a task by the processor CPU. In addition, in the process initiation unit PIU an upwardly counting, n-bit-wide counter C.sub.UP is provided, whose content is added to the content of the t-bits of an initiation cell IC, before this cell is entered into the contention unit PCU, in order to generate an absolute time information that indicates by what time, reckoned from its initiation, this task is to be processed by the processor at the latest. The counter C.sub.UP is clocked by an external clock CLOCK.

Detailed Description Text (9):
In the following, typical method sequences of the above-specified apparatus are explained in more detail with reference to FIG. 2, on the one hand for case A), in which an external event signals the processing of a task, and on the other hand for the case B), in which a subtask is generated by the processor. The numberings indicated in brackets in the following correspond to the numbers indicated in FIG. 2.

Detailed Description Text (10):
A) After the application of an external event, e.g. event e, to one of the signal lines provided therefor of the process initiation unit PIU, the following operational steps are carried out:

Detailed Description Text (17):
(B) In the case of the presence of an internal event, i.e. if a task is requested by another task, the following steps are carried out:

Detailed Description Text (24):
Since the time at which the processing of a task must be concluded at the latest is already determined during the generation of the contention cell CC for this task, the content of the contention unit PCU changes only if a contention cell CC is newly generated or erased, or if the state of an existing contention cell CC changes. For

example, through an alteration of the f-bits of a contention cell, a suppression or resumption of participation in the comparison of the t-bits by the arbiter A can be effected, or, through an alteration of the t-bits of an existing contention cell CC, the time sequence information of the allocated task can be altered. In other words, the priority allocation to the tasks awaiting processing can be altered only if the content of the contention unit alters. All alterations at the contention unit are realized by the hardware, by means of a write access to the contention unit. That is, a comparison is performed of the t-bits of all occupied cells in the contention unit in order to determine which contention cell contains the smallest time sequence information, and, if warranted, the transmission of an interrupt request to the processor CPU must be carried out only if a write access to the contention unit has taken place. At all other times, the contention unit PCU and the arbiter A can be used for other purposes.

Detailed Description Text (26):
As a reaction to a write signal at the line w of the contention unit, a signal is set to high for a determined time at the line c. The time duration for the high signal is dependent on the signal delay within the arbiter. The signal of the line c is logically connected with the output of a logical function PLA (Program Logical Array) via an AND gate. The function PLA is a logical connection of all f-bits of a contention cell CC, and indicates by means of its output (0 or 1) whether or not this contention cell is currently participating in the comparison of the t-bits by the arbiter. The high signal at c brings it about that the output at the dynamic flip-flops D-FF of the arbiter A acquires only those cells CC of the contention unit that are occupied and whose f-bits (PLA function) allow participation in this comparison. In addition, it can be learned from FIG. 3 that the output can be zero only of that flip-flop D-FF that is allocated to the cell having the smallest time sequence information. The just-specified operating mode of the arbiter serves to determine which of the occupied (r-bit=occupied) and active (PLA function=1) contention cells contains the shortest time sequence information, and thus receives the highest priority according to the EDF principle. This operating mode of the arbiter is called priority allocation for short in the following.

Detailed Description Text (28):
FIG. 4 shows the temporal curves of the different signals of the arbiter A of FIG. 3 for the two above-described modes of operation (priority allocation or, respectively, overloading display). If a write access to one of the contention cells is supposed to take place, the line w is set to high. With the leading edge of the signal at w, the arbiter goes into the operating mode for priority allocation, in which a comparison of the time sequence information takes place only of those cells that are not excluded from this comparison by the result of the PLA function. For this purpose, a high signal is applied to the line c simultaneously with the high signal on the line w, in order to activate the working of the f-bits. This high signal at c is applied for the time duration t, which is the time duration required for a write access to a contention cell plus the signal delay due to the arbiter logic. After a further time period t, i.e. with the trailing edge of the signal at the line c, a signal is set to high at the line a, in order to latch the result of the comparison of the t-bits of all admitted cells, obtained by means of the logical connection shown in FIG. 3, into a respective D-FF of the arbiter. If the result of the logical connection of the flip-flop D-FF of the arbiter A changes, an interrupt request is directed to the processor CPU on the line IRINW. In order to avoid the influencing of the operating mode for the overloading display by the priority allocation, a line b is simultaneously (i.e., likewise at the leading edge of the high signal at the line c and for the time duration 2t) set to low, which blocks the clock that latched the result of the comparator COMP into a flip-flop RS-FF provided for the overloading display. After expiration of the time period t, this blocking is removed by the line b.

Detailed Description Text (29):
At the leading edge of the signal at the line b, the arbiter A again enters the operating mode for the overloading display, in which a periodic comparison of all occupied cells of the contention unit takes place independently of the state of the cells indicated by the f-bits (PLA function). This is realized by means of the signal low at the line c. The blocking of the clock with which the result of the comparator COMP (high or low) is latched into the flip-flop RS-FF is removed by

means of a high signal at the line b. In order to prevent this operation mode from influencing the priority allocation, it is ensured by means of a low signal at the line a that in the overloading display operating mode no output of the arbiter logic can be latched into the flip-flops D-FF.

Detailed Description Text (31):
If, in the above-specified operating mode of priority allocation of the arbiter A, it is determined during a comparison of the time sequence information that the just-determined contention cell with the smallest time sequence information is different than in the previously executed comparison, i.e. if a new winner cell CC of the contention unit PCU has been determined, the following further steps are carried out (cf. FIG. 2):

Detailed Description Text (48):
A plurality of f-bits is provided in each contention cell CC in order to enable an intertask communication, such as for example semaphore, event flags or message boxes. A logical function PLA is defined by the f-bits, whose result (0 or 1) indicates whether or not the allocated cell of the contention unit PCU participates in the comparison of the t-bits by the arbiter A. In the following, procedural sequences are explained in more detail by means of the example of a semaphore and message box.

Detailed Description Text (55):
An advantage of the apparatus of the invention is thus, among other things, that the apparatus PICU also takes over the management of semaphores and mailboxes, in addition to the dynamic priority allocation and overloading display. In this way, the operating system can be further relieved, and the utilization of processor capacity increased.

Other Reference Publication (1):
Prozebrechner: statische und terminorientierte Prioritaten im Vergleich--Elektron, 20 (1978), H.6.S. 283-289.

CLAIMS:

1. A method for processing of a plurality of tasks by a processor of a real-time data processing apparatus, in which each task is to be processed individually in a processor according to a dynamically allocated priority and within a predetermined time period, comprising the steps of:

for each task to be processed by the processor, generating an initiation cell having a plurality of t-bits that contain a time sequence information which indicates a time within which a processing of the respective task must be carried out, and a plurality of i-bits that contain an index information referring to a task control block in the data processing apparatus;

in order to log tasks on for processing, entering each initiation cell which has been produced independently from its priority to a free contention cell in a contention unit which is independent of the processor and which has a plurality of contention cells;

for defining a task which has to be executed next, after the entry of the initiation cells comparing the time sequence information in each of the initiation cells entered into the free contention cells in the contention unit in order to determine which task has a lowest time sequence information; and

sending an interrupt signal to the processor in order to initiate processing of the determined task having the lowest time sequence information, provided that the determined task does not correspond with a previously determined task which is currently being processed by the processor.

2. The method according to claim 1 wherein the initiation cell is generated

by a process initiation unit, containing an initiation cell, having t-bits pre-coded or initiated by the processor, for each admitted external event that can signal a

task for processing by the processor.

10. An apparatus for processing of a plurality of tasks within a predetermined time period in a processor according to a dynamically allocated priority, comprising:

means for generating for each task to be processed by the processor an initiation cell having a plurality of t-bits that contain a time sequence information which indicates a time within which a processing of the respective task must be carried out, and a plurality of i-bits that contain an index information referring to a task control block in the data processing apparatus;

means for entering each initiation cell which has been produced independently from its priority to a free contention cell in a contention unit independent of the processor and having a plurality of contention cells in order to log on tasks for processing;

means for comparing, after the entry of the initiation cells, the time sequence information in each of the initiation cells entered into the free contention cells in the contention unit in order to determine which task has a lowest time sequence information in order to define a task which has to be executed next; and

means for sending an interrupt signal to the processor in order to initiate processing of the determined task having the lowest time sequence information, provided that the determined task does not correspond with a previously determined task which is currently being processed by the processor.

L20: Entry 5 of 23 File: USPT Aug 15, 2000

DOCUMENT-IDENTIFIER: US 6105048 A
TITLE: Apparatus and method for the real-time processing of a plurality of tasks

Abstract Text (1):
An apparatus and a method for the processing of a plurality of tasks by a processor of a real-time data processing installation, in which each task is dynamically allocated a priority according to its urgency, after which the tasks are processed by a processor. For this purpose, for each task to be processed, the generation of an initiation cell having a number of t- and i-bits is provided, as is the entering of this initiation cell into a free cell of a contention unit. The time sequence information contained in the t-bits of all cells of the contention unit are compared with one another in order to determine the task containing the smallest time sequence information. In case the determined task does not agree with the previously determined task, an interrupt signal is sent to the processor in order to introduce the processing of this newly determined task.

Brief Summary Text (2):
The present invention relates to an apparatus and a method for the processing of a plurality of tasks in a processor of a real-time data processing installation in which each task is to be processed individually by the processor according to a dynamically allocated priority and within a predetermined time period.

Brief Summary Text (3):
In a real-time system of the type named above having a dynamic priority allocation, the priority of a task or of an interrupt is assigned according to the relative urgency, in relation to the time at which a task is available for processing or an interrupt is requested.

Brief Summary Text (4):
In many of the previously used systems, a static priority allocation is used for tasks and interrupt requests that does not change over the course of time. In order to enable the timely processing of all tasks in such a system with n independent periodic tasks, the utilization of processor capacity may not exceed the theoretical value of n(2.sup.1/n -1). For large n, this value converges rapidly to ln2=0.69. That is, the utilization of processor capacity in such a system can be a maximum of only 70%, and the processor requires at least 30% idle time in order to be able to process all tasks in a timely fashion.

Brief Summary Text (5):
For the improvement of the utilization of the processor capacity, and thus of the performance of the data processing apparatus as a whole, the realization of a dynamic priority allocation has already been tried. A theoretical foundation for such a dynamic priority allocation, with which a processor capacity utilization of 100% can be achieved, can be found in the article entitled "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," in: Journal of the Association for Computing Machinery, vol. 20, January 1973, pp. 46-61. An algorithm for a priority allocation designated Earliest Deadline First (EDF) is therein described. According to EDF, of all the tasks awaiting processing, the one for which the time period for its processing expires the earliest receives the highest priority. In the article named, among other things there is also a comparison of the theoretical performance of a system having a static priority allocation with one having a dynamic priority allocation, and also a comparison of these systems with a

mixed priority allocation.

Brief Summary Text (6):
Although with a dynamic priority allocation according to EDF a utilization of processor capacity of 100% can be achieved, the performance of the system is not improved, since the processor must additionally process the algorithm for the priority allocation. In practice, it has turned out that in some cases the algorithm for the priority allocation causes an overhead of up to 80% of the processor time, so that only 20% of the processor time is available for the actual processing of the tasks.

Brief Summary Text (7):
A reduction of the utilization of processor capacity by the processing of the priority allocation algorithm can be prevented by providing an additional processor for processing the priority allocation algorithm, the additional processor being exclusively responsible for this task. A solution variation of this type can be found in the article "Implementation and Evaluation of a Time-Driven Scheduling Processor," in: EEE Transactions on Computers, 7/88, p. 172ff.

Brief Summary Text (10):
It is an object of the present invention to create a method and an apparatus in which the utilization of processor capacities with the use of a dynamic priority allocation according to EDF, in an apparatus without an additional processor, can be substantially increased in relation to known systems. Such a solution should also be able to be realized without high expense.

Brief Summary Text (11):
The above task is solved, according to the invention, by means of a method for processing a plurality of tasks by a processor of a data processing apparatus, in which each task is to be processed according to a dynamically allocated priority individually and within a predetermined time period. The method comprises the following steps:

Brief Summary Text (14):
Likewise, the posed task is solved by means of an apparatus for the processing of a plurality of tasks by a processor within a predetermined time period according to a dynamically allocated priority, in which apparatus, for an external task to be processed, means are provided for the generation of an initiation cell, consisting of at least a plurality of t-bits, which contain a time sequence information, and of a plurality of i-bits, which contain an index information. The apparatus additionally comprises means for entering the initiation cell of each task to be processed into a contention unit (PCU), in which a plurality of contention cells is provided for this purpose, and also comprises means for the comparison of the time information contained in the t-bits of all cells of the contention unit, in order to determine the task that contains the earliest expiring time information. Means are also provided for the transmission of an interrupt signal to the processor, in order to introduce the processing of the newly determined task, in case the determined task does not agree with the previously determined task just being processed by the processor. The time sequence information contained in the t-bits indicates, in the standard way, the time within which the processing of the task has to occur, while the index information contained in the i-bits refers to the task control block of the data processing apparatus.

Brief Summary Text (16):
improve the utilization of the capacity of the processor, since the processor does not have to process any overhead code for the priority allocation. The processor can thus be loaded with tasks up to the theoretical limit of 100%. The hardware expense for the contention unit is extremely low in comparison with an additional processor. A contention unit can, for example, comprise a simple static RAM memory block or simple binary counters, in which the contention cells are stored.

Brief Summary Text (17):
In an embodiment of the method or of the apparatus, the initiation cell is advantageously generated by means of a process initiation unit, in which, for each admitted external event that triggers a task for processing by the processor, an

initiation cell with t- and i-bits that are pre-coded or initiated by the processor is contained. It is further provided within the scope of the invention that for an internal task to be processed, e.g. a subtask, the t- and i-bits are generated for a contention cell by the processor and are entered into the contention unit.

Brief Summary Text (19):
In addition, it has proven advantageous if each cell of the contention unit additionally contains f-bits that are provided for an intertask communication, e.g. semaphore, event flags or message boxes, by means of which the participation of a contention cell in the comparison of the time sequence information contained in the t-bits can be temporarily suppressed. If a task must await a particular precondition, an f-bit is set in the contention cell of this task, in order to exclude it temporarily from the comparison of the time sequence information contained in the t-bits. If the preconditions for the further processing of these tasks are fulfilled, the f-bit of the allocated contention cell is reset, so that this cell can subsequently take part again in the comparison of the t-bits. In this way, an active and an inactive state of a contention cell can be represented in a simple way.

Detailed Description Text (3):
A plurality of signal lines are connected to the process initiation unit PIU, which respectively represent one external event by means of which a particular task can be signaled for processing by the processor CPU (here not shown). For this purpose, the process initiation unit comprises for each external event signal respectively one initiation cell IC (Initiation Cell), which are all combined into an array ICA (Initiation Cell Array). Each initiation cell IC contains: a determinate number of bits, namely a number n of t-bits, which indicate a relative time information for the corresponding task, within which time interval the task is to be processed at the latest by the processor CPU in order to satisfy the requirements of the real-time processing; a plurality of i-bits, which form an index that refers to the task control block TCB of the data processing apparatus, where further information for the processing of the respective task is stored; and a plurality of f-bits, which are used as flags, e.g. for inter-task communication. The t-bits can already be pre-coded for each possible task in the array ICA or can be written in the array ICA at the startup of the data processing equipment. The i-bits are written in the allocated cell in the initiation phase of a task by the processor CPU. In addition, in the process initiation unit PIU an upwardly counting, n-bit-wide counter C.sub.UP is provided, whose content is added to the content of the t-bits of an initiation cell IC, before this cell is entered into the contention unit PCU, in order to generate an absolute time information that indicates by what time, reckoned from its initiation, this task is to be processed by the processor at the latest. The counter C.sub.UP is clocked by an external clock CLOCK.

Detailed Description Text (9):
In the following, typical method sequences of the above-specified apparatus are explained in more detail with reference to FIG. 2, on the one hand for case A), in which an external event signals the processing of a task, and on the other hand for the case B), in which a subtask is generated by the processor. The numberings indicated in brackets in the following correspond to the numbers indicated in FIG. 2.

Detailed Description Text (10):
A) After the application of an external event, e.g. event e, to one of the signal lines provided therefor of the process initiation unit PIU, the following operational steps are carried out:

Detailed Description Text (17):
(B) In the case of the presence of an internal event, i.e. if a task is requested by another task, the following steps are carried out:

Detailed Description Text (24):
Since the time at which the processing of a task must be concluded at the latest is already determined during the generation of the contention cell CC for this task, the content of the contention unit PCU changes only if a contention cell CC is newly generated or erased, or if the state of an existing contention cell CC changes. For

example, through an alteration of the f-bits of a contention cell, a suppression or resumption of participation in the comparison of the t-bits by the arbiter A can be effected, or, through an alteration of the t-bits of an existing contention cell CC, the time sequence information of the allocated task can be altered. In other words, the priority allocation to the tasks awaiting processing can be altered only if the content of the contention unit alters. All alterations at the contention unit are realized by the hardware, by means of a write access to the contention unit. That is, a comparison is performed of the t-bits of all occupied cells in the contention unit in order to determine which contention cell contains the smallest time sequence information, and, if warranted, the transmission of an interrupt request to the processor CPU must be carried out only if a write access to the contention unit has taken place. At all other times, the contention unit PCU and the arbiter A can be used for other purposes.

Detailed Description Text (26):
As a reaction to a write signal at the line w of the contention unit, a signal is set to high for a determined time at the line c. The time duration for the high signal is dependent on the signal delay within the arbiter. The signal of the line c is logically connected with the output of a logical function PLA (Program Logical Array) via an AND gate. The function PLA is a logical connection of all f-bits of a contention cell CC, and indicates by means of its output (0 or 1) whether or not this contention cell is currently participating in the comparison of the t-bits by the arbiter. The high signal at c brings it about that the output at the dynamic flip-flops D-FF of the arbiter A acquires only those cells CC of the contention unit that are occupied and whose f-bits (PLA function) allow participation in this comparison. In addition, it can be learned from FIG. 3 that the output can be zero only of that flip-flop D-FF that is allocated to the cell having the smallest time sequence information. The just-specified operating mode of the arbiter serves to determine which of the occupied (r-bit=occupied) and active (PLA function=1) contention cells contains the shortest time sequence information, and thus receives the highest priority according to the EDF principle. This operating mode of the arbiter is called priority allocation for short in the following.

Detailed Description Text (28):
FIG. 4 shows the temporal curves of the different signals of the arbiter A of FIG. 3 for the two above-described modes of operation (priority allocation or, respectively, overloading display). If a write access to one of the contention cells is supposed to take place, the line w is set to high. With the leading edge of the signal at w, the arbiter goes into the operating mode for priority allocation, in which a comparison of the time sequence information takes place only of those cells that are not excluded from this comparison by the result of the PLA function. For this purpose, a high signal is applied to the line c simultaneously with the high signal on the line w, in order to activate the working of the f-bits. This high signal at c is applied for the time duration t, which is the time duration required for a write access to a contention cell plus the signal delay due to the arbiter logic. After a further time period t, i.e. with the trailing edge of the signal at the line c, a signal is set to high at the line a, in order to latch the result of the comparison of the t-bits of all admitted cells, obtained by means of the logical connection shown in FIG. 3, into a respective D-FF of the arbiter. If the result of the logical connection of the flip-flop D-FF of the arbiter A changes, an interrupt request is directed to the processor CPU on the line IRINW. In order to avoid the influencing of the operating mode for the overloading display by the priority allocation, a line b is simultaneously (i.e., likewise at the leading edge of the high signal at the line c and for the time duration 2t) set to low, which blocks the clock that latched the result of the comparator COMP into a flip-flop RS-FF provided for the overloading display. After expiration of the time period t, this blocking is removed by the line b.

Detailed Description Text (29):
At the leading edge of the signal at the line b, the arbiter A again enters the operating mode for the overloading display, in which a periodic comparison of all occupied cells of the contention unit takes place independently of the state of the cells indicated by the f-bits (PLA function). This is realized by means of the signal low at the line c. The blocking of the clock with which the result of the comparator COMP (high or low) is latched into the flip-flop RS-FF is removed by

means of a high signal at the line b. In order to prevent this operation mode from influencing the priority allocation, it is ensured by means of a low signal at the line a that in the overloading display operating mode no output of the arbiter logic can be latched into the flip-flops D-FF.

Detailed Description Text (31):
If, in the above-specified operating mode of priority allocation of the arbiter A, it is determined during a comparison of the time sequence information that the just-determined contention cell with the smallest time sequence information is different than in the previously executed comparison, i.e. if a new winner cell CC of the contention unit PCU has been determined, the following further steps are carried out (cf. FIG. 2):

Detailed Description Text (48):
A plurality of f-bits is provided in each contention cell CC in order to enable an intertask communication, such as for example semaphore, event flags or message boxes. A logical function PLA is defined by the f-bits, whose result (0 or 1) indicates whether or not the allocated cell of the contention unit PCU participates in the comparison of the t-bits by the arbiter A. In the following, procedural sequences are explained in more detail by means of the example of a semaphore and message box.

Detailed Description Text (55):
An advantage of the apparatus of the invention is thus, among other things, that the apparatus PICU also takes over the management of semaphores and mailboxes, in addition to the dynamic priority allocation and overloading display. In this way, the operating system can be further relieved, and the utilization of processor capacity increased.

Other Reference Publication (1):
Prozebrechner: statische und terminorientierte Prioritaten im Vergleich--Elektron, 20 (1978), H.6.S. 283-289.

CLAIMS:

1. A method for processing of a plurality of tasks by a processor of a real-time data processing apparatus, in which each task is to be processed individually in a processor according to a dynamically allocated priority and within a predetermined time period, comprising the steps of:

for each task to be processed by the processor, generating an initiation cell having a plurality of t-bits that contain a time sequence information which indicates a time within which a processing of the respective task must be carried out, and a plurality of i-bits that contain an index information referring to a task control block in the data processing apparatus;

in order to log tasks on for processing, entering each initiation cell which has been produced independently from its priority to a free contention cell in a contention unit which is independent of the processor and which has a plurality of contention cells;

for defining a task which has to be executed next, after the entry of the initiation cells comparing the time sequence information in each of the initiation cells entered into the free contention cells in the contention unit in order to determine which task has a lowest time sequence information; and

sending an interrupt signal to the processor in order to initiate processing of the determined task having the lowest time sequence information, provided that the determined task does not correspond with a previously determined task which is currently being processed by the processor.

2. The method according to claim 1 wherein the initiation cell is generated

by a process initiation unit, containing an initiation cell, having t-bits pre-coded or initiated by the processor, for each admitted external event that can signal a

task for processing by the processor.

10. An apparatus for processing of a plurality of tasks within a predetermined time period in a processor according to a dynamically allocated priority, comprising:

means for generating for each task to be processed by the processor an initiation cell having a plurality of t-bits that contain a time sequence information which indicates a time within which a processing of the respective task must be carried out, and a plurality of i-bits that contain an index information referring to a task control block in the data processing apparatus;

means for entering each initiation cell which has been produced independently from its priority to a free contention cell in a contention unit independent of the processor and having a plurality of contention cells in order to log on tasks for processing;

means for comparing, after the entry of the initiation cells, the time sequence information in each of the initiation cells entered into the free contention cells in the contention unit in order to determine which task has a lowest time sequence information in order to define a task which has to be executed next; and

means for sending an interrupt signal to the processor in order to initiate processing of the determined task having the lowest time sequence information, provided that the determined task does not correspond with a previously determined task which is currently being processed by the processor.

Refine Search

Search Results -

Term	Documents
(7 AND 11 AND 13 AND 8).USPT.	4
(L13 AND L11 AND L8 AND L7).USPT.	4

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L14

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Saturday, February 28, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L14	L13 and l11 and l8 and l7	4	L14
L13	L12 or l5	4985	L13
L12	response near1 l3	786	L12
L11	priorit$ near1 l3	1822	L11
L10	pririt$ near1 l3	0	L10
L9	prioit$ near1 l3	0	L9
L8	task$1	160018	L8
L7	work$ thread$	949	L7
L6	work thread$	221	L6
L5	message near3 l3	4425	L5
L4	L3 and l2	1	L4
L3	queue$1	28279	L3
L2	5515538.pn.	1	L2

h e b b cg b e e ch

L1	6006247.pn.	1	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

First Hit Fwd Refs
End of Result Set

Generate Collection Print

L14: Entry 4 of 4 File: USPT Jul 4, 1995

DOCUMENT-IDENTIFIER: US 5430850 A
TITLE: Data processing system with synchronization coprocessor for multiple threads

Brief Summary Text (6):
Parallel programs contain synchronization events. It is well known that processor utilization suffers if it busy-waits; to avoid this, some form of multiplexing amongst threads (tasks or processes) is necessary. This is true even in uniprocessors.

Detailed Description Text (11):
In addition to its conventional RISC instructions, the Data Processor can execute a few additional "dataflow instructions" whose effect is to send messages into the network through a queue 27. These are non-blocking sends, i.e., the Data Processor continues executing after sending a message. The message can cause threads to be scheduled on the other nodes or on the same node, and a later response may deposit values in the sender's frame. As discussed below, by including a message formatter in the Synchronization Processor, the Data Processor can be a fully conventional RISC processor.

Detailed Description Text (14):
The Start Processor has a program counter called SIP ("Start processor Instruction Pointer"), two special registers SFP and SV and, perhaps, other general purpose registers. The Start Processor is triggered by the arrival of a start message from a queue. It simply waits if there is no start message available. When it picks up a start message, its SIP, SFP and SV registers are loaded with values from the message, after which it begins executing instructions from the address in SIP. It can read and write local memory 36 and it can post new thread identifying (FP, L.sub.D) pairs to a queue 32 to be picked up by the Data Processor in initiating new threads of computation. FP is a frame pointer and L.sub.D is a pointer to the block of code for the thread.

Detailed Description Text (37):
Readers familiar with dataflow literature will recognize that the input queue of start messages for the Start Processor to which SFP and L.sub.D are posted corresponds to the "token queue" of dataflow architectures.

Detailed Description Text (51):
Note that if the location is full, an msg.sub.-- rIload message behaves just like an msg.sub.-- rload message. Otherwise, the message information is queued there to be handled later, in response to an msg.sub.-- rlstore message:

Detailed Description Text (57):
Note that if the location is full, an msg.sub.-- rtake message returns the value just line an msg.sub.-- rload message, but it also resets the location to the empty state. Otherwise, the message information is queued there to be handled later, just like a msg.sub.-- rIload message.

Detailed Description Text (106):

h e b b g e e e f c e c b e ge

In the SPMD model, there is no direct access from the i'th node to data in the j'th node, where i.noteq.j. Instead, the program in the i'th node communicates with program in the j'th node using send and receive primitives. This is modelled easily in the *T abstract machine as follows: A part of the global data area in each node is reserved for a message queue, representing messages entering that node. A source-level send command from any node to the node J is compiled by appending a reference to the message data structure into the j'th message queue. Implementing such shared message queues is easily done, using the rput and rget instructions. A source-level receive command in the j'th node is compiled by dequeing a reference to a message data structure its local message queue, and then using rload's to fetch the contents of the message.

Detailed Description Text (177):
Execution continues in this fashion until all the queues of the registered frames are empty. When the M88110 executes a next instruction under this condition, it is given an "out of work" thread which, presumably, deregisters the least recently used frame and registers a frame from the software managed queue of ready frames.

Other Reference Publication (2):
Efficient Dynamic Scheduling of Medium-Grained Tasks for General Purposing Parallel Processing, Musciano et al. Aug. 15, 1988 pp. 166-175.

CLAIMS:

11. A data processing system as claimed in claim 10 wherein the synchronization coprocessor comprises a frame registry which identifies active frames to be given priority in the queue.

L14: Entry 1 of 4 File: USPT Apr 8, 2003

DOCUMENT-IDENTIFIER: US 6546425 B1
TITLE: Method and apparatus for providing mobile and other intermittent connectivity in a computing environment

Abstract Text (1):
A seamless solution transparently addresses the characteristics of nomadic systems, and enables existing network applications to run reliably in mobile environments. The solution extends the enterprise network, letting network managers provide mobile users with easy access to the same applications as stationary users without sacrificing reliability or centralized management. The solution combines advantages of existing wire-line network standards with emerging mobile standards to create a solution that works with existing network applications. A Mobility Management Server coupled to the mobile network maintains the state of each of any number of Mobile End Systems and handles the complex session management required to maintain persistent connections to the network and to other peer processes. If a Mobile End System becomes unreachable, suspends, or changes network address (e.g., due to roaming from one network interconnect to another), the Mobility Management Server maintains the connection to the associated peer task--allowing the Mobile End System to maintain a continuous connection even though it may temporarily lose contact with its network medium. In one example, Mobility Management Server communicates with Mobile End Systems using Remote Procedure Call and Internet Mobility Protocols.

Brief Summary Text (30):
A presently preferred exemplary embodiment of the present invention also allows a system administrator to manage consumption of network resources. For example, the system administrator can place controls on Mobile End Systems, the Mobility Management Server, or both. Such controls can be for the purpose, for example, of managing allocation of network bandwidth or other resources, or they may be related to security issues. It may be most efficient to perform management tasks at the client side for clients with lots of resources. However, thin clients don't have many resources to spare, so it may not be practical to burden them with additional code and processes for performing policy management. Accordingly, it may be most practical to perform or share such policy management functions for thin clients at a centralized point such as the Mobility Management Server. Since the Mobility Management Server proxies the distinct data streams of the Mobile End Systems, it provides a central point from which to conduct policy management. Moreover, the Mobility Management Server provides the opportunity to perform policy management of Mobile End Systems on a per user and/or per device basis. Since the Mobility Management Server is proxying on a per user basis, it has the ability to control and limit each user's access to network resources on a per-user basis as well as on a per-device basis.

Drawing Description Text (11):
FIG. 7 is a diagram showing how a received work request can be dispatched onto different priority queues;

Drawing Description Text (12):
FIGS. 8 and 9 show processing of the contents of the different priority queues;

Detailed Description Text (23):
Referring once again to FIG. 2, Mobility Management Server 102 includes an address translator 220 that intercepts messages to/from Mobile End Systems 104 via a conventional network interface driver 222. For example, address translator 230 recognizes messages from an associated session peer (Fixed End System 110) destined for the Mobile End System 104 virtual address. These incoming Mobile End System messages are provided to proxy server 224, which then maps the virtual address and message to previously queued transactions and then forwards the responses back to the current point of presence addresses being used by the associated Mobile End System 104.

Detailed Description Text (37):
The Internet Mobility Protocol engine 244' then formulates the received message into a RPC receive indication system work request 354, and provides the Mobility Management Server 102 RPC engine 240' with the formulated work request and association-specific context information. When RPC protocol engine 240' receives work request 352, it places it into an association-specific work queue 356, and schedules the association to run by providing a scheduled request to a global queue 358. The main work thread of RPC engine 240' is then signaled that work is available. Once the main thread is awake, it polls the global queue 358 to find the previously queued association scheduled event. It then de-queues the event and begins to process the association-specific work queue 356.

Detailed Description Text (43):
If the type of work request is a "schedule request" (decision block 360), the RPC engine 240' determines which association is being scheduled (block 362). RPC engine 240' can determine this information from what is stored on global queue 358. Once the association is known, RPC engine 240' can identify the particular one of association work queues 356(1) . . . 356(n) the corresponding request is stored on. RPC engine 362 retrieves the corresponding association control block (block 362), and calls a Process Association Work task 364 to begin processing the work in a specific association's work queue 356 as previously noted.

Detailed Description Text (44):
FIG. 5 shows example steps performed by the "process association work" task 364 of FIG. 4. Once the specific association has been determined, this "process association work" task 364 is called to process the work that resides in the corresponding association work queue 356. If the de-queued work request (block 390) is an RPC receive request (decision block 392), it is sent to the RPC parser to be processed (block 394). Otherwise, if the de-queued work request is a pending receive request (decision block 396), the RPC engine 240' requests TDI 204' to start receiving data on behalf of the application's connection (block 398). If the de-queued work request is a pending connect request (decision block 400), RPC engine 240' requests TDI 204' to issue an application specified TCP (or other transport protocol) connect request (block 402). It then waits for a response from the TDI layer 204'. Once the request is completed by TDI 204', its status is determined and then reported back to the original requesting entity. As a performance measure, RPC engine 240' may decide to retry the connect request process some number of times by placing the request back on the associations-specific work queue (356) before actually reporting an error back to the requesting peer. This again is done in an effort to reduce network bandwidth and processing consumption.

Detailed Description Text (63):
Referring once again to FIG. 5, once the "process association work" task 364 process has completed executing its scheduled amount of association work (decision block 404), it checks to see if the dispatch queues require servicing (block 406). FIG. 8 is a flowchart of example steps performed by the "process dispatch queues" block 406 of FIG. 5 to process the dispatch queues 510 shown in FIG. 7.

Detailed Description Text (64):
In this example, dispatch queues 510 are processed beginning with the highest priority queue (510(1) in this example) (block 408). Each queue 510 is assigned a weight factor. The weight factor is a configuration parameter that is returned by the configuration manager 228 when a Mobile End System 104 to Mobility Management Server 102 association is created. As one example, low priority dispatch queues 510 can have a weight factor of 4, and medium priority queues can have a weight factor of 8. High priority RPC calls do not, in this example, use weight factors because they are executed immediately as they are parsed.

Detailed Description Text (66):
If, after exiting the loop, the queue still has work remaining (decision block 418), the queue will be marked as eligible to run again (block 420). By exiting the loop, the system yields the processor to the next lower priority queue (blocks 424, 410). This ensures that all priority levels are given an opportunity to run no matter how much work exists in any particular queue. The system gets the next queue to service, and iterates the process until all queues have been processed. At the end of processing all queues, the system tests to see if any queues have been marked as eligible to run--and if so, the association is scheduled to run again by posting a schedule request to the global work queue. The association is scheduled to run again in the "process global work" routine shown in FIG. 4 above. This approach yields the processor to allow other associations that have work to process an opportunity run. By assigning each queue a weight factor, the system may be tuned to allow different priority levels unequal access to the Mobility Management Server 102's CPU. Thus, higher priority queues are not only executed first, but may also be tuned to allow greater access to the CPU.

Detailed Description Text (70):
FIG. 9 shows how the RPC engine 240' handles proxy server-generated RPC calls. For high priority address and connection objects, the RPC engine 240' dispatches a send request to the Internet Mobility Protocol engine 244' immediately. The send request results in forwarding the RPC message to the peer Mobile End System 104. For lower priority objects, the Internet Mobility Protocol engine 244 send request is posted to an appropriate priority queue 510'. If the association is not scheduled to run, a schedule request is also posted to the global queue 358'. The Internet Mobility Protocol send request is finally executed when the dispatch queues are processed as described earlier in connection with FIGS. 5 & 8.

Detailed Description Text (112):
Work is requested by local clients through the Internet Mobility ProtocolRequestWork( ) function. Once the work is validated and placed on the global work queue, the Internet Mobility ProtocolWorkQueueEligible( ) function is invoked. If in a threaded environment, the Internet Mobility Protocol worker thread is signaled (marked eligible) and control is immediately returned to the calling entity. If in a synchronous environment, the global work queue is immediately run to process any work that was requested. Both methods end up executing the Internet Mobility ProtocolProcessWork( ) function. This is the main dispatching function for processing work.

CLAIMS:

40. A server as in claim 39 wherein said session manager includes a session priority queue that provides at least one configurable session priority for said session.

60. A mobile computing device as in claim 59 wherein said mobile interceptor includes a session priority queue that provides at least one configurable session priority.

L12: Entry 17 of 30 File: USPT Nov 7, 1995

DOCUMENT-IDENTIFIER: US 5465335 A
TITLE: Hardware-configured operating system kernel having a parallel-searchable event queue for a multitasking processor

Abstract Text (1):
A multitasking data processing system is provided with a hardware-configured operating system kernel. The system includes a processor queue that includes a plurality of word stores, each word store storing a task name, in execution priority order, that is ready for processing. An event queue in the kernel includes a plurality of word stores for storing task names that await the occurrence of an event to be placed in the processor queue. When an associated processor signals the occurrence of an event, matching logic searches all word stores in the event queue, in parallel, to find a task associated with the signalled event and then transfers the task to the processor queue. Shift logic is also provided for simultaneously transferring a plurality of task names, in parallel, in the processor queue to make room for a task name transferred from the event queue.

Brief Summary Text (4):
Real-time data processors are able to handle the processing of a number of tasks on a concurrent basis. Multiprocessors perform this function by providing a number of processors that operate on the tasks in parallel. Single processor systems, operating in real-time, handle "parallel" processing on a multitasking basis. Multitasking is a process of executing several tasks concurrently or in parallel. The concurrently processed tasks execute logically at the same time, even though they may not execute physically at the same time.

Brief Summary Text (5):
Multitasking operations perform under control of the computer's operating system and, in particular, under control of an "Executive" segment thereof that controls the scheduling of the concurrently running tasks. The Executive segment (hereinafter called EXEC) provides an interface between the tasks and the central processing unit. To each task, the EXEC appears as the task's own central processing unit. The EXEC handles all of the details involved in sharing the physical CPU with all of the tasks in the system. Thus, an EXEC controls which task should have possession of the CPU at any time by examining priority and readiness levels assigned to each task.

Brief Summary Text (6):
In general, the EXEC enables the running of one task on the CPU until another request to use the CPU is received from a higher priority task. The running task is "suspended", and the higher priority task is allowed to run. This process may occur many times over but sooner or later the highest priority task will complete its processing and voluntarily suspend itself. Thus, even though a lower priority task may be suspended many times before its completion, it eventually completes and produces the same result as if it had run uninterrupted.

Brief Summary Text (7):
Tasks have various execution states. An active task is one which has control of the CPU and is executing. Only one task can be active at any given time on a multitasking CPU. An inactive task is not executing and is not waiting to execute. It is simply idle. A ready task is one which is waiting for CPU time to become available so that it can execute. A waiting task is one that has suspended operation until the occurrence of some event. The event can be generated by another task, or a

hardware event. Upon occurrence of the event, the task is moved from the waiting state to either the ready or the active state depending upon the priority of currently active tasks.

Brief Summary Text (8):
Tasks may be synchronized by priority or by task readiness or a combination of both. In general, an EXEC includes a number of utility routines that are used by tasks to perform necessary control functions. Under these utility routines, any task may schedule another task, suspend itself or another task, signal an event, wait for an event or delay an event for a period of time. The principle EXEC utilities are as follows: Schedule, Suspend, Signal, Wait and Delay. The Schedule utility is used by an active task when it wants another task to begin or resume execution. The Schedule utility allows the highest priority ready task to execute. The Suspend utility is used by an active task to remove itself from the ready state or move another task to the inactive state. Control is then given to the current highest-priority ready task.

Brief Summary Text (9):
The Signal utility is used by the active task or an interrupt service routine to generate a particular event. If no task is waiting on the event that is signalled, the EXEC returns control to the active task. If another task is waiting on the signalled event, control returns to the highest priority task.

Brief Summary Text (10):
The Wait utility is used when the active task wishes to suspend execution and resume on an occurrence of a particular event, or events.

Brief Summary Text (12):
An operating system incorporates an interrupt facility which is the primary means for synchronizing firmware operations with hardware events. When an interrupt occurs, control is transferred from whatever task is executing to an interrupt handling module. One interrupt module exists for each type of interrupt that can occur.

Brief Summary Text (13):
A variety of data structures are employed by the EXEC to implement the utilities above described. A CPU queue is a list of tasks in the ready state. The highest priority task is the currently active task. If there are two ready tasks of the same priority, the task that is ready first becomes the active task. An Event queue is a list of tasks in a waiting state. A Delay queue is another list of delayed events which, after the duration of a delay, will cause tasks waiting on these events to be moved to the CPU queue to become ready tasks.

Brief Summary Text (18):
It is another object of this invention to provide a multitasking data processing system with a hardware implemented operating system kernel wherein queues maintained in the kernel require no separate priority indication.

Brief Summary Text (20):
A multitasking data processing system is provided with a hardware-configured operating system kernel. The system includes a processor queue that includes a plurality of word stores, each word store storing a task name, in execution priority order, that is ready for processing. An event queue in the kernel includes a plurality of word stores for storing task names that await the occurrence of an event to be placed in the processor queue. When an associated processor signals the occurrence of an event, matching logic searches all word stores in the event queue, in parallel, to find a task associated with the signalled event and then transfers the task to the processor queue. Shift logic is also provided for simultaneously transferring a plurality of task names, in parallel, in the processor queue to make room for a task name transferred from the event queue.

Drawing Description Text (7):
FIG. 6 is as block diagram of a set of event count registers used with the invention.

Drawing Description Text (8):
FIG. 7 is a block diagram of a delay timer used with the invention.

Detailed Description Text (2):
In brief, this invention removes certain critical operating system functions from operating system software and implements them in a hardware structure. Through this mechanism, certain often-found data reorganization operations may be performed in parallel and in a predetermined time interval. The hardware kernel employs, in the main, five commands, i.e., Schedule, Suspend, Signal, Wait and Delay. Through these commands, and combinations thereof, a set of real-time operating system primitives are created which enable substantial improvement in an attached processor's operating parameters. Due to the hardware configuration of the queues in the kernel, queue-wide operations are easily accomplished in a fraction of the time needed for similar operations in a software environment.

Detailed Description Text (4):
Turning now to FIG. 1, a high level block diagram is shown of the system. A microprocessor 10 has incorporated therewith, an EXEC module 12 which is implemented in hardware. EXEC module 12 contains three main functional blocks, i.e., a command and status register set 14, a queue state machine 16 and a queue system 18. Queue system 18 contains the data structures that all utilities manipulate. There are three physically identical queues in queue system 18, i.e., CPU queue 20, Delay queue 22 and Event queue 24.

Detailed Description Text (5):
CPU queue 20 is a priority-ordered list of names of Task Control Blocks (TCBs) and associated events that cause the TCBs to be placed in the CPU queue. CPU queue 20 controls the sequence that the various tasks execute within microprocessor 10. TCBs contain all the information pertaining to the status of a task and are used within microprocessor 10 to control such tasks. The name of each TCB is a value (e.g., from 0 to 255) which indicates its priority as well as designating the TCB. The priority of the task, referenced in the task's TCB name, is used to determine when a task is given possession of microprocessor 10. The TCB at the "head" of CPU Queue 20 retains possession of microprocessor 10 for that task. If a task of higher priority is placed in the queue, the currently running task is replaced by the higher priority task at the head of CPU Queue 20 and the task of higher priority executes. At any time, the number of tasks' TCBs in CPU Queue 20 may range from all (all tasks ready to execute), to none (no active or ready tasks).

Detailed Description Text (6):
Event queue 24 contains a priority-ordered list of names of TCBs, with each TCB in the list joined to an event name, upon whose occurrence, the TCB will be moved to CPU Queue 20. Delay Queue 22 contains a list of event names, each event name associated with a delay which indicates the amount of time before the associated event will be signalled. The event names in Delay Queue 22 are prioritized, based upon their associated delay values.

Detailed Description Text (7):
Queue system 18 also includes a set of event count registers 26 that keep track of the availability of a predetermined number of possible events that can occur within the system. It is to be recalled that an event can be signaled and waited on by system tasks. The number of events that can be accommodated by the system is dictated by the size of an address and is not to be considered a limitation of the system. For purposes of description it will be assumed that the system can accommodate up to 256 possible events and utilizes an 8-bit address for any such event.

Detailed Description Text (9):
Queue state machine 16 controls the actions of EXEC module 12. Upon receipt from microprocessor 10 of an EXEC command into a register in command and status registers 14, queue state machine 16 executes the required actions on a queue or queues in queue system 18. Queue state machine 16 also performs required actions on registers in command and status registers 14 or, further, controls a timer 28 and a port control module 30. Port control module 30 provides I/O command functions for EXEC module 12. Address and data connections between microprocessor 10 and EXEC module 12

are provided directly through command and status registers module 14 via lines 31 and 33, respectively.

Detailed Description Text (18):
Commands can be sent to a particular queue to execute arithmetic comparisons. These arithmetic comparisons are done on all queue elements in parallel. The result of such comparisons is used to make parallel shifts in the queue in order to insert or delete items. This yields a significant speed improvement over manipulating these data as is typically done with pointers into a list. The following additional signals run from Queue State Machine 16 to the Event Count Registers 26 over Bus 32 (not specifically shown in FIG. 1)

Detailed Description Text (19):
ECOP (0-1)--Event Counter Operations

Detailed Description Text (20):
ECOC--Event Counter Output Control

Detailed Description Text (21):
ECCLK--Event Counter Clock

Detailed Description Text (22):
ECLR--Event Register Array Clear

Detailed Description Text (23):
EOC--Event Register Output Control

Detailed Description Text (24):
These signals are used to control 256 8-bit registers in Event count registers 26 and a counter therein which can be loaded with any of these values and incremented or decremented.

Detailed Description Text (25):
Each register location in Event Count registers 26 is associated with a correspondingly numbered event. When a "Signal of a particular event occurs, the appropriate register value is read, incremented, and re-written. Likewise, when a Wait is executed on a particular event, the appropriate register's value is read, decremented, and re-written.

Detailed Description Text (27):
Turning now to FIG. 2, the basic structure of CPU queue 20 is illustrated. It will be recalled, that the hardware structure of each queue is identical. TCB names are stored in CPU queue 20 in priority order. Rather than a assigning a separate priority value to a TCB name, it has been determined that substantial storage area can be conserved by assigning as a TCB name, the actual priority value assigned to the TCB. Thus, a TCB having the name 0 has the highest priority value and is referred to by an address indication in EXEC module 12 by an all-0s address. Other TCBs of lower priority are similarly denoted.

Detailed Description Text (32):
Each queue element 35 contains two words of information pertaining to a task. In general, queue element 35 at position 1 in CPU queue 20 will contain two words pertaining to the highest-priority task awaiting action. Those words are the TCB name having the lowest numerical value (priority) and the name of the event that caused the TCB name to be moved into CPU queue 20. Queue elements at position 3, 4 etc. will contain TCB names with lower priority (and higher numerical value).

Detailed Description Text (37):
The Task name/priority is manifest by a TCB name whose value is directly related to its priority (as above described). An "Event Case" is the event name value that occurred that caused the task denoted by a TCB in word A, to be moved into CPU queue 20. An "Event Name" is a name or value given to a specific action within microprocessor 10. For instance, an event name may be a value assigned to a hardware interrupt, an I/O interrupt, etc. A "Delay Value" is a value assigned to a time before an event is to occur.

Detailed Description Text (68):
EVENT COUNT REGISTERS

Detailed Description Text (69):
EXEC module 12 includes 256 event count registers 26, one for each possible event (as limited by an 8-bit address). In FIG. 6, an 8-bit event counter 100 and the first three event count registers 102, 104 and 106 are shown. Event counter 100 is programmable and responds to either an event count being loaded via data bus D(0-7) or to an event count from one of registers 102, 104 or 106.

Detailed Description Text (70):
Each event count register has an assigned value indicative of one of 256 events which can occur in the system. If a register indicates a plus count, that is an indication that multiple tasks are waiting for the event to occur. If the event count register indicates a negative value, the indication is that the event has been signalled (occurred) more times than there are tasks waiting for the event's occurrence. Upon the occurrence of an event, the event count register corresponding to that event is examined to see the state of its count. If the count is seen to be positive, then queue state machine 16 knows that a task is present in event queue 24 and is awaiting the occurrence of the specific event. In such a case, event queue 24 is searched, in parallel, to find all TCBs that specify the specific event. The highest priority TCB that specifies the event is then chosen for execution. Queue state machine 16 transfers the chosen TCB from event queue 24 to CPU queue 20, where it is placed in priority order. When the task is removed from event queue 24, the value in the event count register is decremented through the action of counter 100.

Detailed Description Text (71):
When an event occurs, the Signal utility causes the value of the corresponding event count register to be read and used to program counter 100. Then, counter 100 decrements the value, which decremented value is then written back into an appropriate register. A Wait utility causes the same sequence to occur with the exception that the value is decremented. The value of any event count register (102,104, 106, etc.) is available to be read by queue state machines 16 via data bus D(0-7). The following command lines from queue state machine 16 are applied to counter 100.

Detailed Description Text (75):
ECLR is used to initialize all event count registers 102, 104, 106, etc. to 0.

Detailed Description Text (76):
DELAY TIMER

Detailed Description Text (77):
In FIG. 7, the details of delay timer 28 in FIG. 1 are shown. Delay timer 28 counts system clock cycles and causes queue state machine 16 to signal any event whose delay is up. The number of clock cycles in a delay unit (i.e. a unit of delay time) is programmable and is held in a timer interval register within command and status registers 14. The delay unit value comes into an 8-bit counter 110 in delay timer 28 on TIR(0-7) lines from the timer interval register. 8-bit counter 112 counts the number of delay units by accumulating the number of clock cycles in each delay unit and then incrementing to a next delay unit count. 8-bit counter 112 counts up to 255 and then rolls over to 0. The Delay utility delays events by a relative time, not an absolute time.

Detailed Description Text (81):
Schedule Register (a Write Register): When a Task TCB is written to this register, the TCB is placed in CPU Queue 20 according to its priority (TCB value).

Detailed Description Text (82):
Suspend Register (a Write Register): When a Task TCB is written to this register, the TCB is removed from CPU Queue 20, if it is indeed in CPU Queue 20. If the TCB is not in CPU Queue 20, an interrupt condition is generated to microprocessor 10. Status indicating this condition is set in a Status Register.

Detailed Description Text (83):
Signal Register (a Write Register): When an Event control block is written to this register from microprocessor 10, this event is signalled. If no task is waiting for this event, the only action taken is to decrement the Event Count for this event. If tasks are waiting for this event, then the TCB with the highest priority (lowest TCB value) is removed from Event Queue 24 and placed in CPU Queue 20.

Detailed Description Text (84):
Wait Register (a Write Register): When an Event control block is written to this register from microprocessor 10, the running task's TCB is placed in Event Queue 24 to wait for the named event. The running task's TCB is found at the first queue position in CPU Queue 20 (position 0). This TCB is removed from CPU Queue 20 and placed in Event Queue 24 according to its priority (TCB value).

Detailed Description Text (85):
Delay Register (a Write Register): When an Event control block and an 8-bit delay value is written to this register from microprocessor 10, the Event control block name is placed into Delay Queue 22, prioritized by its delay value. The lower the delay, the closer to the head of the queue this block name is placed.

Detailed Description Text (87):
Timer Interval Register (a Write Register): The user writes a 16-bit value to this register which represents the number of system clocks that make up each delay value.

Detailed Description Text (88):
CPUQ Register, Event Q Register, Delay Q Register (Read Registers): These registers, when read by microprocessor 10, will sequentially give the contents of all the Queue Element Words in the appropriate queue. These are Diagnostic Registers.

Detailed Description Text (90):
Active TCB Register (a Read Register): When read, the TCBName of the current task that has highest priority in the CPU Queue is returned.

Detailed Description Text (91):
Event Case Register (a Read Register): When read, the Event control block Name of the event that caused a task to resume execution is returned. This value only has meaning if this task has executed a Wait utility call.

Detailed Description Text (95):
Assume that microprocessor 10 writes a TCB of a task that it wishes to schedule, to the Schedule Register within command and status registers 14 in EXEC module 12. The TCB value is thereby latched within the Schedule register. In response, queue state machine 16 performs a parallel search of CPU queue 20 to find a stored next lower priority TCB from the TCB being scheduled. Once found, the TCBs within CPU 20 are shifted one position to the right, starting from the incoming TCB's queue position. The position within CPU 20 vacated by this rightward shift is loaded with the TCB of the task to be scheduled.

Detailed Description Text (97):
CPU queue 20 has now been modified so that the TCB name of the task to be scheduled is present in CPU queue 20 in its proper priority position. Microprocessor 10, to determine what is now the highest-priority task awaiting execution, reads the Active TCB register from command and status registers 14. Microprocessor 10 may also read the Event case register. The access by microprocessor 10 to the Active TCB register causes queue state machine 16 to read the TCB name from the first queue element (highest priority) in CPU queue 20 and to place this value onto microprocessor data lines DATA (0-15). Port control 30 then asserts the NDTACK line to indicate that the value of the active TCB register is available to microprocessor 10.

Detailed Description Text (98):
It will be recalled that the Event case is the name of the event that caused a name of a task to be moved into CPU queue 20. Microprocessor 10 can access the Event case register to determine the event value within CPU queue 20. This access causes the event name to be read from the first queue element in CPU queue 20 into the Event

case register and thence to be placed on microprocessor 10's data bus DATA(0-15). The signal NDTACK is asserted indicating to microprocessor 10 that the value of this register is available on the output data bus.

Detailed Description Text (100):
The following are states that occur during a Schedule utility. Inputs to Queue state machine 16 are in lower case and outputs are uppercase. All events are synchronized to the system clock. For each state that is not exited until a particular event occurs, the event is indicated. If no event is specified for a given state, the state is exited upon the next state machine clock.

Detailed Description Text (101):
1. IDLE--EXEC chip select (cs) is not asserted, ntst is not asserted, nrst is not asserted, timer carry out (tco) is not asserted. Event--Chip select (cs) is asserted, CPU wishes to make an operating system call.

Detailed Description Text (105):
5. READ MATCH>ADD--Read address lines for highest priority match>address. Actions--Store the value on ADDR(0-7) in a temporary register, and de-assert the MATCH line.

Detailed Description Text (110):
10. SETUP END uP XACTION (Setup to end transaction with CPU. Actions--DTACK (data transfer acknowledge) is asserted. Event--cs is de-asserted.

Detailed Description Text (113):
At this time, the task's TCB has been placed in the CPU Queue. In order to use the updated queue information, microprocessor 10 must execute a "READ.sub.13 ACTIVETCB" command. In addition, microprocessor 10 may execute a "READ.sub.13 EVENTCASE" in order to determine the event that caused this task to resume execution. These actions are performed for any utility call including Schedule, Suspend, Signal, Wait, and Delay.

Detailed Description Text (115):
1. IDLE--EXEC chip select (cs), ntst, nrst, and tco (timer carry out) all are de-asserted.

Detailed Description Text (117):
3. WRITE ACTIVETCB REG--Write the Active TCB Register with the TCBName at the head of the CPU Queue. Actions--The Active TCB Register is written with the value now appearing on the D(0-7) lines. This is the TCBName of the next task to have use of the CPU resource by virtue of its priority.

Detailed Description Text (118):
4. SETUP END uP XACTION--Setup to end transaction with CPU. Actions--DTACK (data transfer acknowledge) is asserted. Event--cs is de-asserted.

Detailed Description Text (122):
1. IDLE--EXEC chip select (cs), ntst, nrst, and tco (timer carry out) all are de-asserted.

Detailed Description Text (125):
4. SETUP END uP XACTION--Setup to end transaction with CPU. Actions--DTACK (data transfer acknowledge) is asserted. Event--cs is de-asserted.

Detailed Description Paragraph Table (1):
Inputs to EXEC Module 12 CS - chip select R/W - read/not (write) line uPAddr (0-7) - address lines SCLK - EXEC system clock (not shown) NRST - External Reset line NTST - Test line (for test mode operation) Bidirectional lines to EXEC Module 12 uP Data (0-15) - data lines Outputs from EXEC Module 12 NDTACK - data transfer acknowledge (asynchronous acknowledge) NINT - interrupt line to CPU

Detailed Description Paragraph Table (2):
Queue Name Word A Word B

CPU Queue Task Name/Priority Event Case Event Queue Task Name/Priority Event Name Delay Queue Delay Value Event Name

Other Reference Publication (2):
Test and Evaluation of the SVID-Compliant REAL/IX Real Time Operating System by Zuccarelli et al, IEEE 1990, pp. 81-85.

Other Reference Publication (3):
FASTCHART--A Fast Time Deterministic CPU and Hardware Based Real-Time-Kernel by Lennant Linch et all, IEEE Publication, Jun. 1991, pp. 36-40.

CLAIMS:

1. A multitasking data processing system including a hardware-configured portion of an operating system, the combination comprising:

processor queue means configured in hardware and including a plurality of word stores, for storing in priority order, task names ready for execution;

event queue means configured in hardware and including a plurality of word stores, for storing task names that await an occurrence of an event to be placed in said processor queue means; processor means for signalling occurrence of an event; and

match logic means configured in hardware and responsive to an asynchronously signalled event from said processor means for searching said word stores in parallel in said event queue means to find a task name associated with said signalled event occurrence, and for transferring said task name to said processor queue means.

5. A multitasking data processing system recited in claim 1, further comprising:

hardware event count means, including a plurality of registers, each register storing a count of tasks awaiting said asynchronously signalled event from said processor means; and

search means responsive to said asynchronously signalled event to search said registers in parallel in said hardware event count means to determine whether a task is awaiting said asynchronously signalled event, and if so, operating said match logic means to search said event queue means to find a name of a task awaiting said asynchronously signalled event.

6. A multitasking data processing system recited in claim 5, wherein each said word store in said event queue means stores a task name and an associated event name, an occurrence of said named event causing said named task to be transferred to said processor queue means.

7. A multitasking data processing system recited in claim 6, wherein task names in said event queue means are stored in a priority order assigned to each task, and a parallel search of said event queue means by said match logic means causes a readout of all task names associated with a signalled event, in parallel and in priority order.

8. A multitasking data processing system recited in claim 5, wherein said search means further determines, for a signalled event, if more events have occurred than there are tasks awaiting such event's occurrence.

9. A multitasking data processing system as recited in claim 1 further comprising:

hardware delay queue means including a plurality of word stores, each word store storing a delay interval value and an event to be signalled by said processor means upon occurrence of said delay interval value;

timer means for signalling delay interval values;

means responsive to a delay interval value signalled by said timer means for

searching word stores in said delay queue means, in parallel, to find an event to be signalled upon a signalling of a delay interval value and to transmit said signalled event to said match logic means.

10. A multitasking data processing system as recited in claim 1 wherein each said task name is represented by a unique value, said value also indicating said named task's priority among all tasks.

11. A hardware-configured operating system kernel, comprising:

a set of command and status hardware registers for receiving both addresses and data from a connected data processing system and for passing data to said data processing system;

a CPU queue including a plurality of hardware word stores for storing a queue of task names, said queue of task names organized in priority order of said named tasks, all said tasks being ready for execution;

a hardware-configured event queue including a plurality of word stores for storing a queue of task names in priority order, each said task name stored in association with an event name upon whose occurrence, said associated task name will be ready for execution; and

a queue state machine responsive to asynchronously occurring event data in said command and status registers, to cause a parallel search of said event queue to find all task names associated with said asynchronously occurring event data, and to enable a transfer of said task names to said CPU queue.

12. The kernel as recited in claim 11, wherein said queue state machine causes said task names to be stored in said CPU queue in task name priority order.

13. The kernel as recited in claim 12, wherein each said task name is a unique value, said value indicative of said named task's priority.

14. The kernel as recited in claim 13, further comprising:

a delay queue including a plurality of word stores for storing a queue of delay interval values and associated event names, said event names ordered in said delay queue by increasing value of associated delay interval values, said queue state machine responsive to a delay interval value manifestation to cause an associated event name to be signalled and task name transfers from said event queue to be enabled.

17. The kernel as recited in claim 16, further comprising:

a plurality of event registers, one for each of a plurality of events, each said event register indicating a count of tasks awaiting occurrence of an event, said queue state machine responsive to a signalled event to search all said event registers in parallel to determine if any tasks were awaiting occurrence of said signalled event, and if so, to cause transfer of a highest priority task name associated with said event in said event queue.

L15: Entry 1 of 2 File: USPT Jun 9, 1998

DOCUMENT-IDENTIFIER: US 5764953 A
**** See image for Certificate of Correction ****
TITLE: Computer implemented system for integrating active and simulated decisionmaking processes

Detailed Description Text (15):
The real-time system interface 16 passes the real-time domain events received from SMS interface 34 and user interfaces 36 to a queue 38 associated with event processor module 20, as indicated by line 40. In FIG. 1, the real-time domain events generated by system interface 16 are denoted by the label "R-T EVENT." The queue 38 is provided to linearize the asynchronous real-time events received from system interface 16. By queueing the events, event processor module 20 can process them one-at-a-time without overlap to avoid conflicts in recommendations issued by A/S module 14. The events in queue 38 preferably are priority-ordered based on relative priorities associated with each type of event. The external events having the highest priorities are dequeued from priority-ordered queue 38 first. Events having the same priority may be ordered in a first-in-first-out manner, such that the event received first is dequeued first. The event processor module 20 affixes to each real-time event dequeued from priority-ordered queue 38 a time-stamp reflecting the current time. The real-time clock 26 passes the current time to event processor module 20, as indicated by line 42. The event processor module 20 then records the time-stamped event in an event record stored in real-time event file 32, as indicated by line 44. The real-time event file 32 contains a sequence of time-stamped event records received by event processor module 20 over a period of time, providing a source of captured events for use in the simulation mode.

Detailed Description Text (21):
When the dequeued event is an assignment-event, event processor module 20 updates the resource domain model in storage device 12 by adding the committed call to the schedule of the particular technician. However, event processor module 20 does not request an assignment and scheduling recommendation from A/S module 14 if the assignment-event is consistent with an outstanding recommendation. In this case, the change in the status of the call does not affect the assignment or scheduling of other calls. If the assignment-event overrides a recommendation by A/S module 14, a recommendation may be necessary for affected calls or technicians. When the dequeued event is a clear-call-event, event processor module 20 does not request an assignment and scheduling recommendation, and updates the resource domain model only if the actual completion time of the call is significantly different than the scheduled completion time. The event processor module 20 determines whether the actual and scheduled completion times are significantly different by comparing the difference between the time-stamp of the event and the scheduled completion time to a predetermined time interval. The event processor module 20 ascertains the difference by reference to the schedule contained in the resource domain model in storage device 12.

Detailed Description Text (28):
The event processor module 20 records both the external and internal simulated events as simulated event records in simulated event file 30 for future analysis, as indicated by line 60. The internal events are also passed to a time-ordered queue 58 associated with event processor module 20 for incorporation in the simulation, as indicated by line 62. In FIG. 1, the internal simulated events are denoted by the label "INT SIM EVENT." The time-ordered structure of queue 58 requires that internal

events having the earliest time-stamps are dequeued first. The simulation controller module 22 is responsible for controlling simulation clock 24 and managing time-ordered queue 58. As indicated by line 64 of FIG. 1, event processor module 20 passes the time-stamp of each external event dequeued from priority-ordered queue 38 to simulation controller module 22 as a clock control event. The simulation controller module 22 controls simulated clock 24, as indicated by line 66, by driving simulated time forward based in part on the time-stamps of the clock control events. As a result, the external events from priority-ordered queue 38 and internal events from time-ordered queue 58 can be properly interleaved to support the capability of system 10 to operate the simulation with captured real-time events. As indicated by line 68, simulation controller module 20 manages time-ordered queue 58 by dequeueing internal events when simulated time reaches the time-stamps on the respective events. The dequeued internal events are thus submitted to event processor module 20 for appropriate action, thereby incorporating the internal events in the simulation.

Detailed Description Text (44):
FIG. 4 is a block diagram of a second embodiment of a system for integrated resource assignment and simulation in accordance with the present invention. The system 100 shown in FIG. 4 substantially corresponds to system 10 shown in FIG. 1. For example, the operation of system 10 and system 100 in the real-time mode is identical. In the simulation mode, however, system 100 is configured to control the rate of advance of simulated time in order to accommodate the participation of system users in the simulation mode. Like system 10, system 100 includes a system interface 16 comprising SMS interface 34 and user interfaces 36. In addition to serving as a source of real-time events, user interfaces 36 provide user input events for incorporation in the simulation mode. Specifically, as indicated by line 40, user interfaces 36 pass user input events to priority-ordered queue 38 via system interface 16 to be interleaved with the external simulated events produced by simulated event generator module 18. In FIG. 4, the user input events provided by user interfaces 36 are denoted by the label "UI EVENT."

Detailed Description Text (46):
Each of the user input events is associated with a priority relative to one another and relative to the external simulated events generated by simulated event generator module 18. To enable incorporation of the user input events in the priority-ordered queue 38 during simulation, system 100 incorporates an external-event-input function and a clock-advance function. The external-event-input function, which forms part of the process of simulated event generator module 18, prevents external events from being enqueued on priority-ordered queue 38 until simulated time catches up to the transaction-times of the events. Without the external-event-input function, simulated event generator module 18 would pass all of the events sequentially read from real-time event file 32 into priority-ordered queue 38 as quickly as they can be read. The external-event-input function effectively acts as a throttle, as indicated by symbol 102 of FIG. 4, to limit the number of captured events enqueued in queue 38. This throttle effect ensures that internal events are interleaved with external events at the appropriate simulated time, and reduces the amount of processing required to maintain the priority-ordered structure of queue 38 each time a new event is enqueued. The clock-advance function, which forms part of the process performed by simulation clock 24, uses a clock-limit, clock-increment, and a delay to prevent simulated time from overtaking user input. The clock-advance function slows the advancement of simulation clock 24 to provide an appearance of accelerated, slowed, or real-time performance, enabling user interaction with the simulation. The values of clock-increment and delay are freely-adjustable by the system user to adapt the rate of the simulation to individual needs. As will be described, the effect of the external-event-input and clock-advance function is that the stream of simulated events never overtakes simulated time, no event is handled before its simulated time arrives, and during periods when no simulated events are available, simulated time advances at a controlled rate, rather than simply jumping ahead to the next event.

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L18	L17 and l11	14	L18
L17	l13 and l4	284	L17
L16	L15l13 and l4	0	L16
L15	L14 and l11	2	L15
L14	L13 same l4	53	L14
L13	l3 near1 l1	861	L13
L12	L11 and l9	30	L12
L11	L10 near4 l5	29011	L11
L10	predetermined	1022050	L10
L9	L8 and l1.ab.	147	L9
L8	L7 and l6	1451	L8
L7	interrupt$	288498	L7
L6	l1 and l2 and l3 and l4 and l5	1686	L6
L5	time interval$1	142080	L5
L4	priorit$	119845	L4
L3	real time	88144	L3
L2	timer	111261	L2
L1	event$1	428335	L1

END OF SEARCH HISTORY

L18: Entry 2 of 14 File: USPT Aug 28, 2001

DOCUMENT-IDENTIFIER: US 6282673 B1
TITLE: Method of recording information system events

Parent Case Text (3):
PRIORITY CLAIM

Detailed Description Text (8):
The RTC 221 is integrated in the system recorder 220 on the back plane 204. The RTC 221 comprises two 32-bit counters which keep track of real time and elapsed time in seconds. The RTC 221 comprises a four-byte field (i.e., 32 bits) for recording time for over 125 years (2 32 seconds) without having to reset itself. It is designed to count seconds when its input power (V.sub.cc) is applied and continually count seconds under battery backup regardless of the condition of V.sub.cc. The continuous counter is used to derive time of day, week, month, and year by using a software algorithm. Alternatively, the RTC 221 is used under the control of the system recorder 220 to record real time events. Communication to and from the RTC 221 takes place via a 3-wire serial port. A one byte protocol selects read/write functions, counter clear functions and oscillator trim. The RTC 221 records real time in an absolute format. The O/S uses a reference point in time in order to synchronize the RTC 221 with the standard 24-hour time format.

Detailed Description Text (11):
The back plane 204 monitors a plurality of temperature sensors located on a temperature bus (not shown in this figure) once every predetermined time interval, e.g., every second. Each temperature sensor comprises a transducer connected to and having an address at a serial bus (not shown in this figure) on the back plane 204. These transducers are read in the same sequence as their address order. The temperature may range between -25 and +70 degrees Celsius. If any of the temperature sensors reaches +55 degrees Celsius, or -25 degrees Celsius, then a warning is issued, and a message corresponding to that event is written to the NVRAM 224, and sent to other destinations via the system interface 214 and the remote interface 240. If any of the temperature sensors reaches +70 degrees Celsius, then a shutdown command is typically issued and the system is powered off.